Escola Politécnica da USP
Engenharia Mecatrônica

9,3
(nove e três)

Projeto de Formatura

Sistema ERP Didático baseado em J2EE

Érico Resende Santos

Orientador: Marcos Ribeiro Pereira Barretto

Agradecimentos

Ao professor Marcos Ribeiro Pereira Barretto pelo grande apoio e incentivo.

Aos meus pais e irmãos pelo apoio em toda a minha trajetória até aqui.

Resumo

A proposta desse projeto é a construção do módulo de operações de um sistema ERP com base em tecnologia Java 2 Enterprise Environment. O resultado do projeto foi disponibilizado como um software de código aberto para fins didáticos.

O projeto objetivou desenvolver esse sistema utilizando componentes J2EE (que podem ser utilizados em outros sistemas) tanto para testar e difundir a solução como para oferecer uma alternativa de um módulo simples para planejamento de materiais e capacidade em código aberto.

O resultado é bastante inovador com relação à discussão da plataforma J2EE como uma tecnologia relativamente recente e poderosa para desenvolvimento, implementação e manutenção de software e à modelagem orientada a objetos (e especialmente orientada aos requisitos de projeto) para um sistema MRP/CRP.

Além disso o projeto tem um valor especial no sentido didático (que inclusive consta do seu nome) já que, acreditamos, um estudante que queira se iniciar no uso da plataforma J2EE encontrará no resultado desse projeto fonte valiosa de informação (incluindo diversos exemplos para uma aplicação relativamente complexa).

Abstract

This project proposes the building of an operations module of an ERP system using J2EE technology. The result is an open source software for learning purpose.

The utilization of J2EE components intends to test and disseminate the solution and to deliver a simple alternative to production planning and control as an open source software module.

The result is pretty innovative as it discusses the platform J2EE as a modern technology for software development, deployment and maintenance and the object oriented (and use case driven) modeling for a MRP/CRP system.

The project has also a special value in the didactic sense because, as we believe, a student who is whiling to start using de J2EE platform will find in the result of this project valuable source of information (including plenty of examples for a relatively complex application).

Índice

Índice de Figuras

1 Introdução

A proposta desse projeto é a construção do módulo de operações de um sistema ERP com base em tecnologia Java 2 Enterprise Environment.

O resultado do projeto deve ser disponibilizado como um software de código aberto para fins didáticos.

Existem diversas ferramentas no mercado que lidam com o planejamento de compras e produção utilizando a técnica MRP / MRP II. No entanto essas ferramentas, embora modulares, são fechadas não permitindo um intercâmbio de componentes de software, além de demandarem altos investimentos.

A linguagem de programação java e o ambiente de execução J2EE (Java 2 *Enterprise Environment* – Java 2 Ambiente Empresarial) surgiram no mercado como uma poderosa alternativa para o desenvolvimento de software utilizando em larga escala o conceito de componentização.

Com base nisso, este projeto objetiva desenvolver um sistema utilizando componentes J2EE (que possam ser utilizados em outros sistemas) tanto para testar e difundir a solução como para oferecer uma alternativa de um módulo simples para planejamento de materiais em código aberto.

O relatório está organizado conforme explicado a seguir: O capítulo 2 apresenta os conceitos MRP, MRPII e ERP. O capítulo 3 discute a linguagem Java e a tecnologia J2EE, além de conceitos sobre modelagem orientada a objetos e ciclo de desenvolvimento de software, apresentando uma série de conceitos úteis. O capítulo 4 apresenta o projeto desde a construção do modelo até a implementação. O capítulo 5 apresenta os testes realizados e descreve o processo de implementação do sistema no ambiente de teste. O capítulo 6 traz as conclusões e o 7 traz as referências bibliográficas.

2 MRP, MRPII e ERP

2.1 Introdução

Oden et. al. (1993) apresentam seis estratégias para planejamento e controle da produção:

- *Administração de Projetos*: voltada principalmente para organizações que trabalham com projetos. Pode fazer uso de técnicas como PERT (Program Evaluation and Review Technique – técnica para avaliação e revisão de programas) e CPM (Critical Path Method – método do caminho crítico).
- *MRP, CRP e MRP II*: Diferencia a demanda de itens dependentes e independentes e calcula a demanda de itens dependentes com base no planejamento mestre de produção (ver item 2.2 para maiores detalhes).
- *Just-in-Time*: Nessa estratégia cada estação de trabalho "puxa" a produção da estação imediatamente anterior na linha de produção de acordo com a necessidade, sendo esta ditada pelas necessidades de produtos finais.
- *Controle Contínuo de Processo:* Pode ser descrita como uma hierarquia de funções cujos principais níveis são: controle de entrada/saída e medições; controle regulador, monitoramento de processo e gerenciamento de processo.
- *Sistema de Controle Flexível:* Controla um sistema flexível de manufatura, que incorpora técnicas de M&CRP a outras permitindo o planejamento desde produção em larga escala a produção de itens com alta diferenciação.
- *Sistema de Controle Ágil:* Controla um sistema de manufatura ágil: uma mistura de MRP II e Just-in-Time.

Os autores apresentam também os seguintes processos de manufatura:

- *Projeto*: material, produto e pessoas são alocados em função do produto;
- *Centros de Trabalho:* os recursos de produção são organizados pelo seu tipo em centros de trabalho;
- *Linha de produção em pequenos lotes:* linha de produção para vários tipos de produto sendo produzidos em pequenos lotes;
- *Linha de produção e grandes lotes:* produção de itens com pouca diferenciação em grandes lotes;
- *Linha de produção contínua:* produção continua normalmente utilizada para produção de *commodities* como açúcar, aço etc.;

- *Sistemas flexíveis de manufatura:* sistemas de produção totalmente automatizadas em que todo o processo e controlado por um sistema computacional integrado;
- *Sistemas ágeis de manufatura:* sistema também automatizado mas com uma estratégia diferenciada para responder à demanda da forma mais rápida possível de acordo com o tipo de negócio;

E as seguintes estratégias de resposta à demanda:

- *Design-to-order* – projeto por encomenda: os projetos só são realizados e os produtos necessários comprados mediante a ordem do cliente.
- *Make-to-order* – produçao por encomenda: o projeto e algumas matérias primas estão disponíveis anteriormente à ordem mas toda a produção só é disparada depois que a ordem é recebida.
- *Assemble-to-order – montagem por encomenda*: todos os sub-itens do produto final ficam disponíveis mas a montagem final só é realizada após o recebimento da ordem do cliente.
- *Make-to-stock – produção para estoque*: O produto final é produzido em uma quantidade especificada de forma que supra a demanda por um certo período e fica estocado. Assim todos os pedidos são supridos assim que chegam.
- *Make-to-demand – produção para demanda:* estratégia totalmente focada na demanda que pode estocar determinados itens ou o produto final de acordo com a demanda.

Com essas definições, os autores propõem o seguinte mapa para situar os sistemas de planejamento e controle da produção:

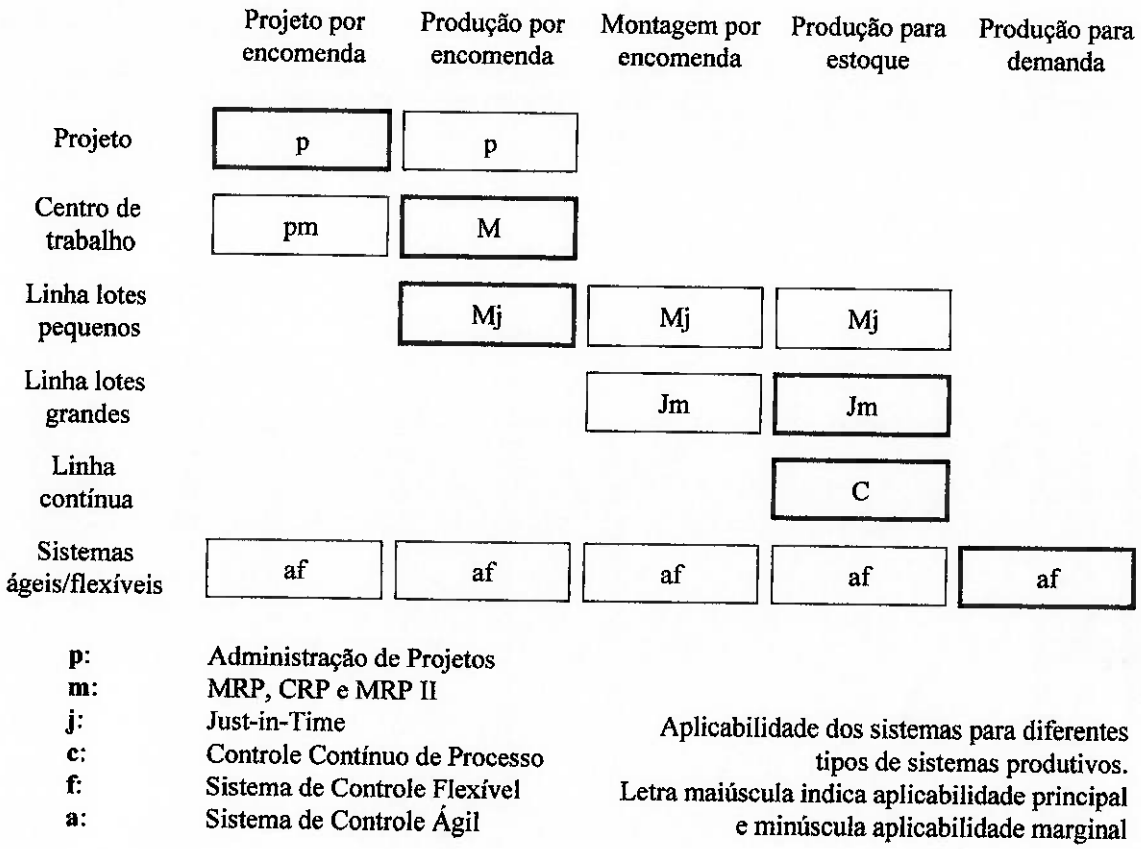


Figura 1. Matriz para seleção de sistemas de planejamento e controle da produção (Oden et. al., 1993).

Percebemos então que a técnica de MRP é apropriada para empresas que utilizam conceito de centros de trabalho ou produção em lotes e que respondem à demanda com produção ou montagem por encomenda e produção para estoque. Ou seja, o MRP/ MRPII é apropriado para um grande número de empresas de manufatura existentes.

2.2 *MRP*

Neste item foram agrupados e estendidos diversos conceitos dos autores Oden et. al. (1993); Corrêa et.al. (1996); Fullmann et.al. (1989); Figueiredo (2001) entre outros.

O MRP (*Material Requirement Planning* ou planejamento de necessidades de materiais) é um sistema baseado em computador para planejamento de estoque de produtos de demanda dependente. Essa técnica passou a ser utilizada a partir dos anos 70 como uma forma de diminuir altas quantidades de estoque e diminuir ao mesmo tempo as freqüentes faltas de produtos nas linhas de montagem. O lema do MRP é: "*ter o material certo no lugar certo no tempo certo*" (Oden et. al. 1993).

Para isso o MRP se vale do conceito de demanda dependente e independente. Os itens efetivamente considerados na previsão de demanda são os itens finais produzidos e efetivamente vendidos pela empresa. Esse itens têm, portanto, demanda independente. No

entanto eles são normalmente compostos por diversos outros itens de demanda dependente. A técnica MRP procura, a partir da previsão de venda dos itens de demanda independente, calcular as necessidades de todos os outros itens de demanda dependente como forma de diminuir ao máximo os estoques desnecessários.

A entrada do MRP é composta pelas necessidades surgidas da demanda (que pode ser gerada pelo módulo MPS), pela capacidade disponível de produção administrada pelo módulo CRP (se o sistema suporta esse módulo) e pela lista hierarquizada de itens de produção (chamada BOM, ou Bill of Materials), além da lista mestre de itens, que contém as características individuais de cada item.

A saída é o conjunto de ordens de compra/produção de todos os itens de demanda dependente no momento oportuno.

2.2.1 Tabela mestre de itens

A tabela mestre lista todos os itens produzidos ou comprados pela firma que participam do processo produtivo. Para cada item, a lista pode trazer as seguintes propriedades.

Código	Identifica o item de forma única.
Descrição	Descrição do item
Unidade	Unidade do item (gramas, ml, unidade etc.)
Custo	Custo unitário do item
Quantidade do lote	Quantidade mínima em que o item é produzido/comprado
Lead time	Tempo até que o produto esteja disponível (compreende compra, transporte, produção etc. dependendo do item)
Estoque de segurança	Estoque mínimo para este item
Produzido/Comprado	Flag que indica se este item é produzido ou comprado
Final/Intermediário	Flag que indica se este item é final ou intermediário
Real/Fantasma	Flag que indica se este item é real ou fantasma (um item fantasma não é estocado. Se trata de uma montagem intermediária).
Calculado com MRP	Indica se este item entra no calculo com MRP. Se não o item tem sua demanda controlada por outro método como ponto de reposição.

Essas características, juntamente com as outras entradas, servem como base para a realização do cálculo do MRP e definição da demanda de cada item individualmente.

2.2.2 BOM – *Bill of Materials*

A Figura 2 mostra um exemplo de lista de materiais para uma lapiseira.

A lista de materiais é definida pela engenharia e detalha como um produto é produzido hierarquicamente, contemplando os itens que o compõe e a quantidade de cada. Através

dela se define a demanda dependente de um componente em função da demanda do item acima na hierarquia do qual esse componente faz parte.

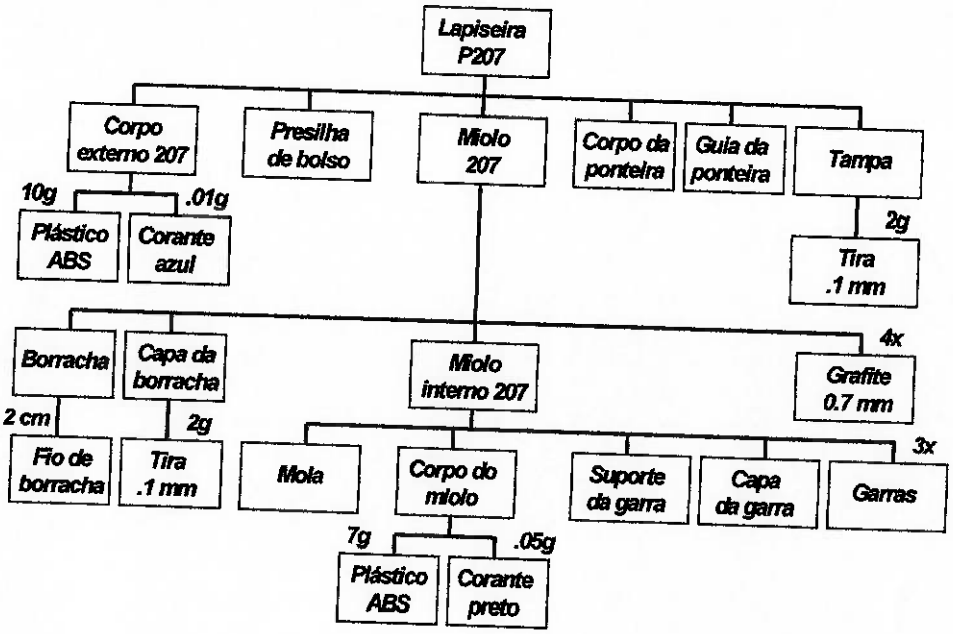


Figura 2. Exemplo de uma lista de materiais (BOM - Bill of Materials)

Cada registro de item da BOM representa uma conexão Pai/Filho e tem as seguintes características:

Código	Identifica um item BOM em uma estrutura unicamente.
Código da estrutura	Identifica uma estrutura (estamos assumindo que um mesmo item final pode ter mais de uma estrutura possível)
Código do item	Código único do item atribuído na tabela mestre.
Código do item Pai	Código único do pai atribuído na tabela mestre.
Descrição	Descrição para essa posição da estrutura.
Quantidade	Quantidade do filho que compõe o pai.
LLC	*Low level code* – trata-se do maior nível (mais baixo na hierarquia) em que este item é encontrado em uma lista de materiais. É necessário para evitar o reprocessamento de um mesmo item várias vezes se este ocorre várias vezes na mesma estrutura e em níveis diferentes.
% perda	Percentagem de perda desse componente na fabricação do item pai nessa estrutura.
Stop	Data de expiração desse registro
Start	Data de início de validade desse registro.

2.2.3 MPS – *Master Production Schedule*

O MPS – *Master Production Schedule* (planejamento mestre da produção) corresponde á previsão de vendas dos itens de demanda independente. Ele é normalmente elaborado pelo departamento comercial da companhia e traz a quantidade necessária de cada item de demanda independente em cada período. O registro do MPS tem os seguintes dados:

Código do item	Código único do item atribuído na tabela mestre.
Data	Data da demanda
Previsão	Previsão de demanda desse item nessa data.
Quantidade disponível	Quantidade já disponível nessa data.
Necessidade líquida	Quantidade necessária a ser produzida para satisfazer a demanda (levando-se em conta o estoque já disponível e o estoque de segurança).

Nesse projeto não será contemplado um módulo específico de MPS e toda a demanda será definida diretamente na tabela de registro de estoque apresentada no próximo item.

2.2.4 Funcionamento do MRP

O cálculo de necessidades de materiais é feito para cada componente considerando-se o *lead time* ou tempo para que o determinado componente esteja disponível, o estoque mínimo (ou estoque de segurança), o estoque inicial e a quantidade mínima do lote, ou a quantidade em que o item pode ser fabricado ou comprado.

Como sugere a Figura 2, o cálculo é feito de forma hierárquica. A demanda informada pelo MPS seria, no caso, de um determinado número de lapiseiras no período. O cálculo do MRP deve gerar, a partir disso, as ordens de compra, montagem ou produção de cada um dos componentes com base no componente logo acima na hierarquia a não ser que estes sejam administrados de outra forma (ver item 2.2.5).

Nós utilizaremos para o cálculo do MRP a tabela de registro de estoque (exemplo na Tabela 1). A necessidade bruta independente vem diretamente do MPS (Nesse projeto será definida diretamente pelo operador). A necessidade dependente surge da "explosão" da demanda do item pai desse item (o miolo no nosso exemplo). O recebimento programado é atribuído pelo próprio sistema decorrente de uma ordem já liberada para a fabricação/compra do item. As necessidades líquidas e ordens recebidas e liberadas são geradas pelo sistema de acordo com a necessidade.

Para o cálculo do estoque projetado em cada período é utilizada a seguinte expressão:

$$\text{Estoque projetado} = \text{Estoque projetado no período anterior} + \text{Recebimento programado} + \text{Ordens recebidas} - \text{Necessidades brutas dependente e independente}$$

Quando o estoque projetado se torna menor que o estoque de segurança, o sistema gera uma necessidade líquida de acordo com a seguinte expressão:

Necessidade líquida	=	Necessidades brutas dependente e independente	+	Estoque de segurança	-	Recebimento programado	-	Estoque projetado no período anterior

Tabela 1. Exemplo de cálculo de MRP com uso de registros de estoque.

Miolo interno | Lead time: 2 | Est. seg.: 0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Nec. Bruta Dep.	25	25	25	35	35	25	20	20	25
Nec. Bruta Indep.			10					5	
Recebimento programado		50	50						
Estoque projetado	0	25	40	5	0	0	0	0	0
Necessidade líquida					30	25	20	25	25
Ordens recebidas					30	25	20	25	25
Ordens a liberar			30	25	20	25	25		

As ordens recebidas somente diferem da necessidade líquida quando há uma política de lotes para o item que não permita um pedido exato. Nesse caso o que sobrar dessa ordem entra como recebimento programado nos próximos períodos.

As ordens a liberar correspondem às recebidas com o período descontado pelo *lead time*.

As ordens a liberar são o resultado final do processamento do MRP. Para que se tornem ordens efetivas é necessário que sejam liberadas por alguém. Nesse caso será criada uma ordem em outro registro que, conforme definido aqui, já estará fora do módulo de MRP.

O cálculo é feito então, a partir do item do MPS (lapiseira, de acordo com a Figura 2), explodindo os itens abaixo deste na árvore e gerando as ordens para cada item em função das necessidades líquidas, tamanho do lote e estoque de segurança. A Figura 3 mostra o fluxograma do funcionamento do MRP.

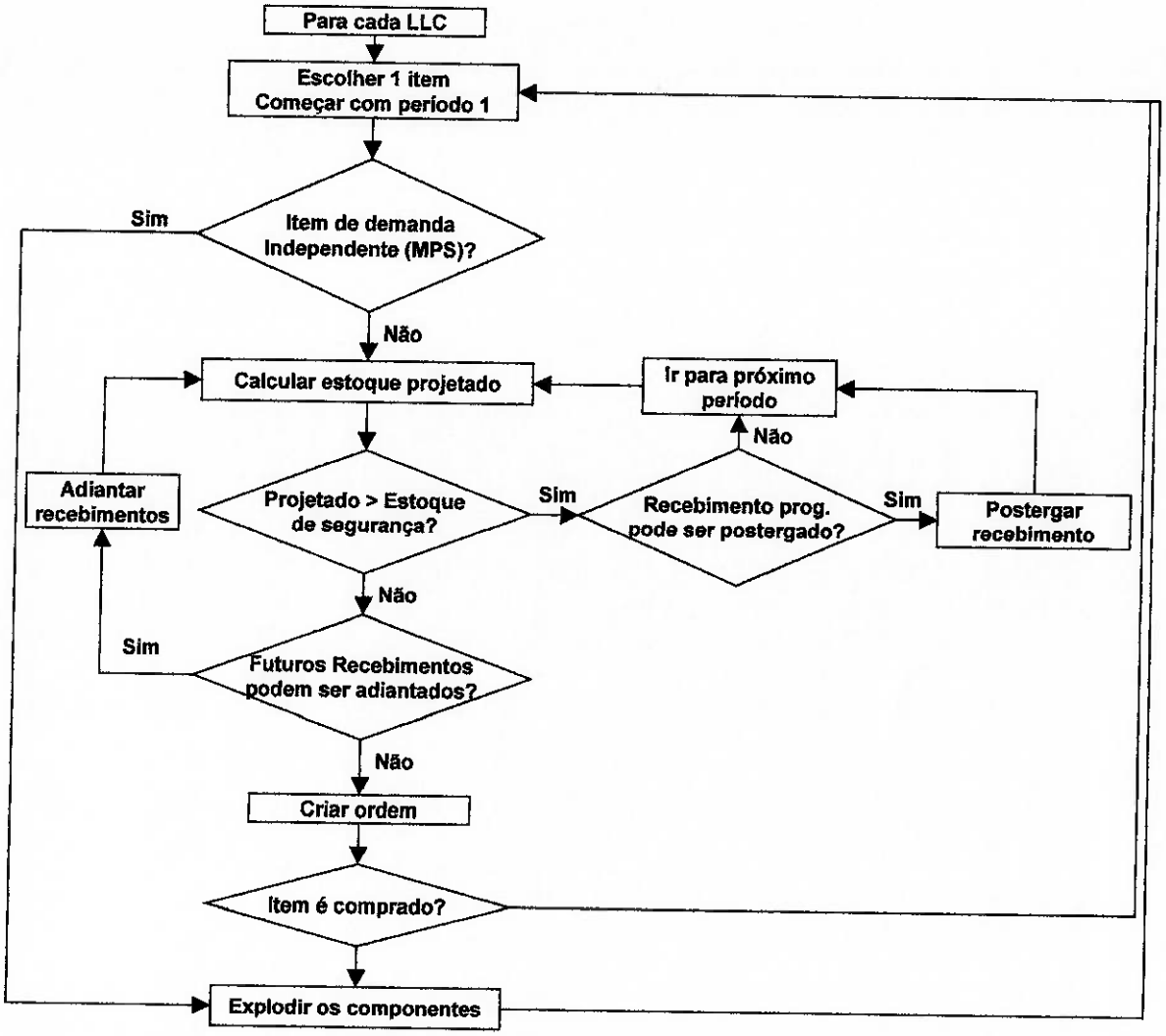


Figura 3. Funcionamento geral do MRP (adaptado de Oden et. al., 1993).

2.2.5 Exceções

Eventualmente alguns itens com grande utilização não são considerados no cálculo por MRP para simplificação e economia de recursos. Exemplos são parafusos, tinta e vários outros que, eventualmente são muito utilizados e comprados constantemente. Para esse tipo de produto normalmente é utilizada outra forma de controle de estoque como o **ponto de reposição**.

Ponto de reposição é um procedimento que define uma quantidade mínima do estoque de determinado produto, levando-se em conta a taxa média de utilização e o tempo necessário para sua aquisição (*lead time*). Assim, quando a quantidade atinge esse nível é liberada uma ordem para mais compra/produção do material.

2.3 CRP

O CRP – *Capacity requirements planning* (planejamento de necessidade de capacidade) surge como complemento ao MRP. O MRP assume, quando calcula todas as necessidades de material, que a capacidade é infinita. O CRP introduz um cálculo das necessidades de recursos de produção de forma que todas as ordens geradas pelo MRP possam ser efetivamente cumpridas.

Sabe-se que o CRP é bem menos utilizado que o MRP pelas empresas em geral. No entanto, em muitos casos, o uso correto dessa ferramenta pode representar uma boa vantagem competitiva para uma empresa, diminuindo estoques de materiais em processo, aumentando a eficiência da fábrica de forma geral entre outras vantagens (Oden et.al., 1993).

A entrada para o CRP é composta pelas ordens a liberar geradas pelo MRP, pelos registros de recursos de produção e pelas informações de processos, além da lista de materiais.

2.3.1 Recursos

Recursos são todas as entidades que realizam algum tipo de atividade no processo produtivo. Isso inclui máquinas (como tornos, fresadeiras, furadeiras, robôs etc.), fornos, transportadoras, operadores etc.

Um registro típico de recurso contém os seguintes dados:

Código do recurso	Identifica o recurso de forma única
Descrição	Descrição do recurso
Tipo	Tipo do recurso (torno, operador etc.)
Tempo de setup	Tempo normal de setup para esse recurso (pode ser diferente para cada processo, mas a lista de processo assume esse valor se nenhum outro for fornecido).
Disponibilidade de horas	Quantidade de horas disponível por hora ou semana para cada unidade do recurso

2.3.2 Processos

São as atividades de transformação, montagem, transporte etc. presentes no processo produtivo.

Um processo conecta um item da árvore de materiais que tenha filhos com um recurso definindo quantas unidades do item são produzidas por hora (produtividade) e qual o tempo de setup para a operação. Poderiam ser definidas várias características para o processo como tempo de espera, transporte etc. Para simplificação nosso modelo somente considera a produtividade (que gera uma carga em horas proporcional à quantidade de itens a serem produzidos) e o setup (que gera uma carga fixa por item por período).

2.3.3 Funcionamento do CRP

Tendo as ordens a liberar geradas pelo MRP, os recursos e processos conectados à árvore de materiais, o CRP calcula a demanda de horas necessárias de cada recurso em cada período de acordo com a seguinte expressão:

$$\text{Demanda de horas} = \text{Ordens a liberar} \times \text{Produtividade} + \text{Setup}$$

A demanda total é obtida depois que toda a árvore de materiais foi percorrida e o tempo de cada processo computado.

Essa demanda é então comparada com a capacidade efetiva de cada recurso em cada período de forma a se identificar eventuais carências de capacidade. Isso é feito por meio do registro de capacidade que tem os seguintes dados:

Código do recurso	Código único do recurso atribuído na tabela de recursos.
Data	Data analisada
Necessidade horas	Demanda calculada em horas
Necessidade qtde.	Demanda calculada em quantidade de recursos
Disponibilidade horas	Disponibilidade de horas do recurso nessa data
Disponibilidade qtde.	Quantidade disponível do recurso nessa data
Nec. Líquida horas	Necessidade líquida em horas (demanda - disponível)
Nec. Líquida qtde	Quantidade líquida necessária (demanda - disponível)

Se existir uma necessidade líquida positiva, ou seja, se em determinada data a demanda de horas de determinado recurso é maior que o disponível, o sistema acusa e alguma medida como redistribuir a demanda, aumentar a capacidade ou rever o MPS pode ser tomada:

Se a necessidade líquida for muito negativa, ou seja, a disponibilidade é bem maior que a demanda, podem ser tomadas medidas contrárias como reduzir a capacidade ou aumentar a demanda.

2.4 MRPII

O MRPII (*Manufacturing Requirement Planning* ou planejamento de recursos de manufatura) consiste em uma evolução dos sistemas MRP com a inclusão do CRP e de outros módulos.

De forma geral os sistemas MRPII estão organizados conforme esquematizado na Figura 4, onde cada box corresponde a um módulo. São eles:

- S&OP – Sales & Operations Management (Gestão de vendas e operações): Trata-se de um módulo gerencial onde são definidos os objetivos da organização, os produtos, políticas de vendas etc.

- MPS – Master Production Schedule (Planejamento mestre de produção): Módulo já apresentado anteriormente onde são definidas as vendas de acordo com uma previsão de demanda.
- RCCP – Rough Cut Capacity Planning (Planejamento de capacidade macro): Trata-se de uma avaliação preliminar da capacidade instalada para atender à previsão de demanda obtida do MPS.
- MRP – Material Requirements Planning (Planejamento de requisitos de material): Ver item 2.2.
- CRP – Capacity Requirement Planning (Planejamento de requisitos de capacidade): Ver item 2.3.
- SFC – Shop Floor Control (Controle de chão de fábrica): Trata-se do controle detalhado da produção.
- Compras: Responsável pela aquisição dos itens de acordo com o cálculo do MRP e liberação das ordens.

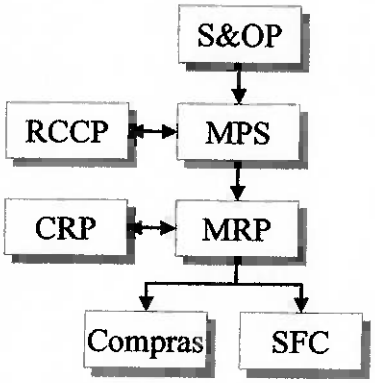


Figura 4. Esquema geral de um sistema MRPII

2.5 ERP

O ERP – *Enterprise Resources Planning* (planejamento de recursos empresariais) é mais uma evolução em relação ao MRP II incorporando módulos para gestão contábil/financeira, RH, contas etc.

A denominação foi criada e popularizada por fabricantes de software que vendem esse tipo de sistema. Normalmente são pacotes robustos compostos por diversos módulos e com base de dados centralizada com objetivo de integrar todas as operações na empresa. Daí serem chamados também de **sistemas integrados de gestão**.

A Figura 5 ilustra como ocorreu a evolução de MRP para MRPII para ERP com a adição de módulos para um controle cada vez mais integrado das operações na empresa.

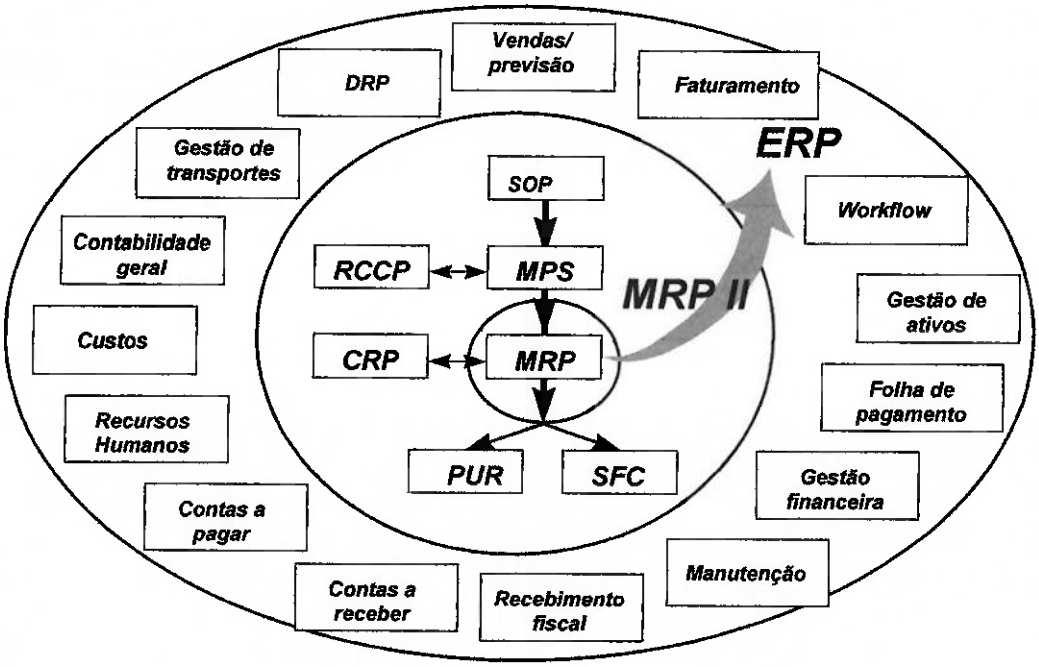


Figura 5. Esquema ilustrativo dos conceitos MRP, MRPII e ERP

3 Linguagem Java e o Ambiente J2EE

3.1 Introdução à Linguagem Java

A tecnologia Java (desenvolvida pela Sun Microsystems®) surgiu como uma resposta poderosa à necessidade de portabilidade de software, ou seja, da necessidade de que um programa desenvolvido pudesse ser utilizado em diferentes tipos de computadores (java.sun.com, 2002). Por possuir essa característica dizemos que a linguagem Java é **multi-plataforma**.

Essa característica permite que código desenvolvido em Java possa ser utilizado em qualquer tipo de computador que possua um ambiente para interpretar esse código. Esse ambiente (comumente chamado ***Java Virtual Machine - JVM***) pode inclusive ser adicionado de forma nativa a outros componentes eletrônicos como eletrodomésticos e automóveis de forma a integrá-los a outros sistemas (Sá, 1999).

Além disso, Java é uma linguagem de programação que nasceu com o objetivo de abordar de forma completa o conceito de orientação a objeto. Isso significa, entre outras coisas, que java consegue **separar a interface da implementação**.

As linguagens tradicionais estruturadas como Fortran, C, Pascal etc. abstraem a solução de um problema em termos da estrutura do computador (que sempre realiza procedimentos de forma seqüencial) enquanto linguagens orientadas a objeto permitem a abstração do problema em termos da estrutura real do problema (Eckel, 2000).

Por fim a linguagem java procura ser o mais **segura** possível no sentido de separar *threads*, contar com o "garbage collector" e manter uma extensa biblioteca de funções já testadas (Roman, 2002).

3.1.1 Características Importantes da linguagem Java

Threading

Em java tudo é um objeto (Eckel, 2000). E todos os objetos java podem possuir capacidade de lidar com paralelismo, ou seja, um programa java torna relativamente fácil a realização de várias tarefas "simultaneamente" com a manipulação das *threads*.

Garbage Collector

A linguagem java foi criada para ser simples e reduzir o tempo de desenvolvimento de software. Por isso o programador não precisa se preocupar em destruir os objetos quando esses não serão mais utilizados. O próprio ambiente de execução conta com um mecanismo chamado *garbage collector* que realiza essa limpeza periodicamente, o que evita erros e falta de memória.

Persistência
Os objetos java podem implementar uma característica comum que os torna persistentes, ou seja, os torna passíveis de serem gravados em um arquivo ou banco de dados quando o programa não está em execução.

Além dessas a linguagem java tem várias outras características que a tornam extremamente vantajosa em vários tipos de aplicação, principalmente em ambientes com processamento distribuído e, especialmente, aplicações que utilizam tecnologia Internet.

Nessa área a linguagem java tem se destacado e se tornou a preferida para grande parte dos fabricantes de software tanto no cliente quanto no servidor. Nós iremos nos concentrar nesse último para a realização desse projeto.

3.2 O Ambiente Tecnológico J2EE

3.2.1 Visão geral

J2EE é um ambiente proposto pela Sun® para implementação de sistemas servidores multi-plataforma, portáveis, seguros e multi-usuários escritos em java (Roman et.al., 2002).

Trata-se na verdade de uma especificação que compreende uma série de bibliotecas de classes e APIs – Application Program Interfaces (interfaces de programação), além das normalmente incluídas na plataforma java padrão (J2SE – Java 2 Standard Edition).

A Figura 6 mostra um esquema genérico da plataforma J2EE. A plataforma genérica deve incluir as seguintes APIs (Roman et.al., 2002).

- **Enterprise JavaBeans (EJB)**: arquitetura para componentes servidores (base da plataforma J2EE).
- **Remote Method Invocation (RMI) e RMI-IIOP**: forma nativa da linguagem java para comunicação entre objetos distribuídos. RMI-IIOP é uma extensão de RMI para uso com CORBA.
- **Java Naming and Directory Interface (JNDI)**: utilizado para acesso a serviços de diretórios.
- **Java Database Connectivity (JDBC)**: interface para acesso a bancos de dados relacionais.
- **Java Transaction API (JTA) Java Transaction Service (JTS)**: especificações para lidar com transações.

- **Java Messaging Service (JMS)**: uma alternativa ao RMI para comunicação entre objetos por meio de mensagens.
- **Java *Servlets***: são componentes java que dão funcionalidade aos servidores Web respondendo a requisições http.
- **Java Server Pages (JSP)**: são parecidos com *servlets* mas não são arquivos com código java puro. São mais fáceis de manter e apropriados para separar o cliente da implementação.
- **Java IDL**: implementação de CORBA baseada em java. Apropriado para lidar com objetos distribuídos utilizando tecnologia CORBA.
- **JavaMail**: serviço que permite o envio de e-mails independentemente de plataforma e de protocolo.
- **J2EE Connector Architecture (JCA)**: permite o acesso a sistemas legados na organização com sistemas em mainframes, ERPs etc. por meio de drivers (que só precisam ser escritos uma vez para cada sistema).
- **Java API for XML Parsing (JAXP)**: para interpretar documentos XML.
- **Java Authentication and Authorization Service (JAAS)**: para realizar operações relacionadas a segurança.

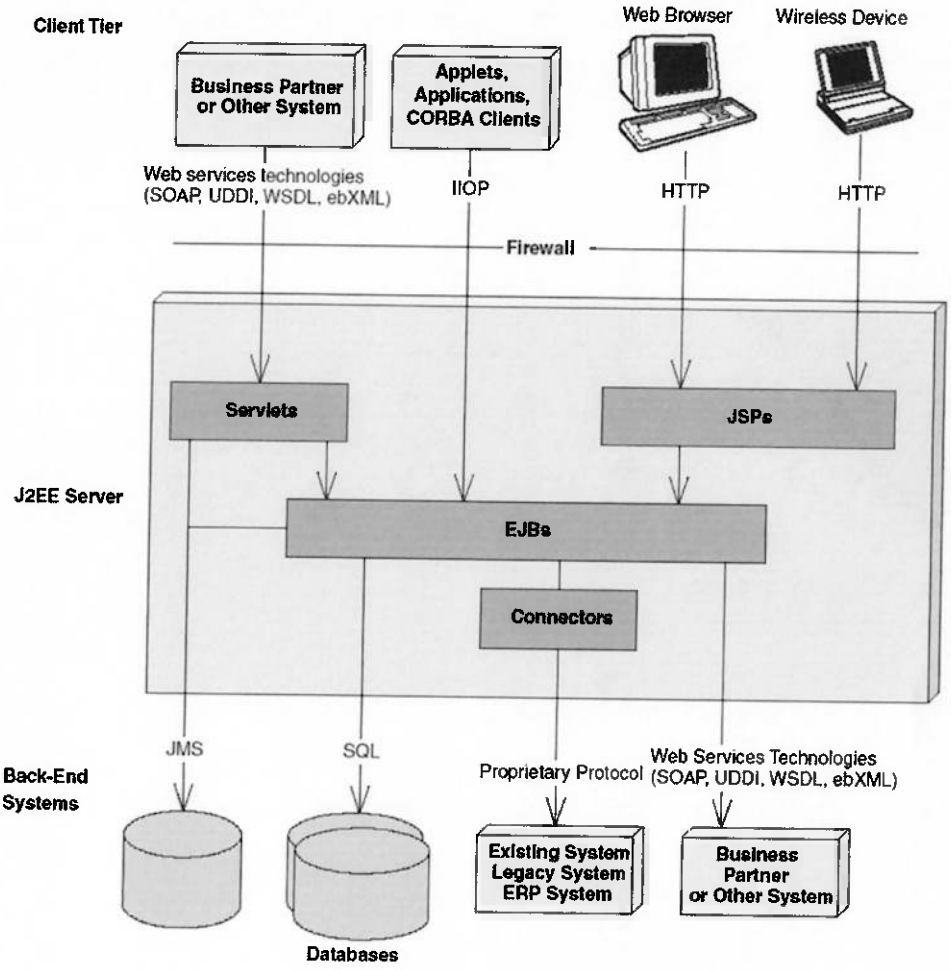


Figura 6. Esquema de implementação genérico da plataforma J2EE (Roman et.al., 2002)

3.2.2 Enterprise JavaBeans (EJB)

Os EJBs são a peça fundamental na plataforma J2EE. Trata-se de uma arquitetura para a construção de componentes servidores.

A idéia de utilização de componentes tem ganho muitos adeptos ultimamente embora não seja adotada em larga escala, principalmente pelos grandes fabricantes de software. No entanto isso tende a mudar (Roman et.al., 2002).

Com a utilização de EJBs, o fabricante do componente, ou da peça de um sistema, tem a vantagem de utilizar uma série de serviços de *middleware* (serviços intermediários entre o sistema desenvolvido e o sistema operacional, rede etc.) pré-definidos e administrados por um servidor de aplicações. Entre os serviços de *middleware* que o servidor de aplicações pode administrar estão:

- Invocação de método remoto (RMI),
- Balanceamento de carga,
- Falha transparente (o cliente é direcionado para outro servidor sem que perceba),

- Integração com sistemas legados,
- Transações (retorno ao estado inicial a não ser que todo o processo seja completado),
- *Caching* (armazenamento de informação no servidor para consulta rápida) etc.

Servidor de aplicações

Para realizar todas essas operações de *middleware*, é necessário um servidor de aplicações (*container*). O servidor de aplicação é responsável por criar e administrar todos os EJBs que fazem parte da aplicação, além de gerenciar os serviços de *middleware* de acordo com o ambiente em que o sistema está implementado.

Exemplos de servidores de aplicação são: Weblogic da BEA, iPlanet da Sun, Websphere da IBM, Oracle 9i, Jrun da Macromedia, Power Tier da Persistence, Gemstone da Brokat, Bluestone da HP, iPortal da IONA, AppServer da Borland e o JBoss de código aberto.

Nesse projeto iremos utilizar o JBoss.

Tipos de EJB

Existem três tipos principais de EJB:

Session bean: modela regras de negócio, ou seja, realiza ações. Os sessions beans funcionam como controles do sistema.

Entity bean: modela dados do negócio, ou seja, são objetos que armazenam dados relativos a alguma entidade.

Message-driven bean: modela ações como os session beans mas só podem ser acessados por mensagens. Funcionam principalmente para a realização de tarefas em lote.

Composição de um EJB

Um EJB é formado por uma série de classes derivadas de interfaces da especificação para EJBs, ou seja, um EJB é, na verdade, formado por vários arquivos. Esses arquivos são:

- A classe Bean, que implementa javax.ejb.SessionBean, javax.ejb.EntityBean ou javax.ejb.MessageDrivenBean,
- O objeto EJB, que herda de javax.ejb.EJBObject,
- O objeto Home, que herda de javax.ejb.EJBHome,
- O objeto EJB local, que herda de javax.ejb.EJBLocalObject (não obrigatório),
- O objeto Home local, que herda de javax.ejb.EJBLocalHome (não obrigatório),
- *Deployment descriptor* (descrição de implementação). Trata-se de um arquivo (XML, no caso do JBoss), que registra os requerimentos de middleware de cada EJB.
- Arquivos específicos de cada fabricante de servidor de aplicação.

Funcionamento do EJB

Um cliente não acessa diretamente a classe Bean. Ele acessa na verdade o objeto EJB que contém uma interface igual à do Bean. Isso ocorre para que os serviços de *middleware* fiquem implícitos e administrados pelo container. Da mesma forma, o cliente acessa o

objeto home para criar um objeto EJB. Assim o objeto home funciona como uma "fábrica" de EJBs.

Tanto o objeto home como o EJB implementam a interface java.rmi.remote que compreende os serviços de comunicação responsáveis por acessar os objetos distribuídos.

As Figura 7 mostra como é feito o acesso do cliente via o objeto EJB. A Figura 8 mostra como funciona o objeto home na criação de um objeto EJB. Pode-se perceber pelas figuras que ambos os objetos implementam a interface remota e podem ser acessados via RMI.

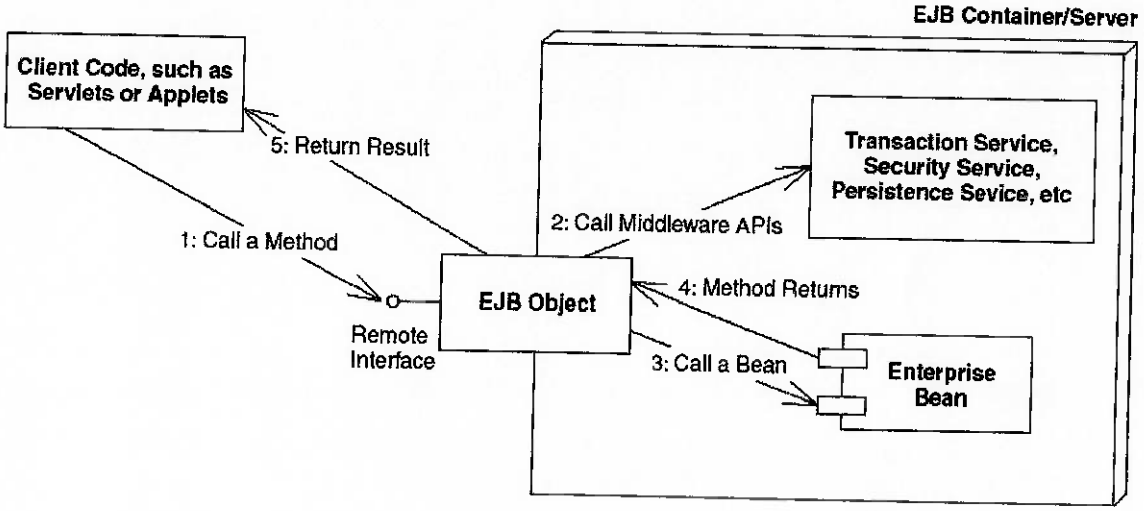


Figura 7. Exemplo de acesso a Enterprise Bean via o objeto EJB (Roman et.al., 2002).

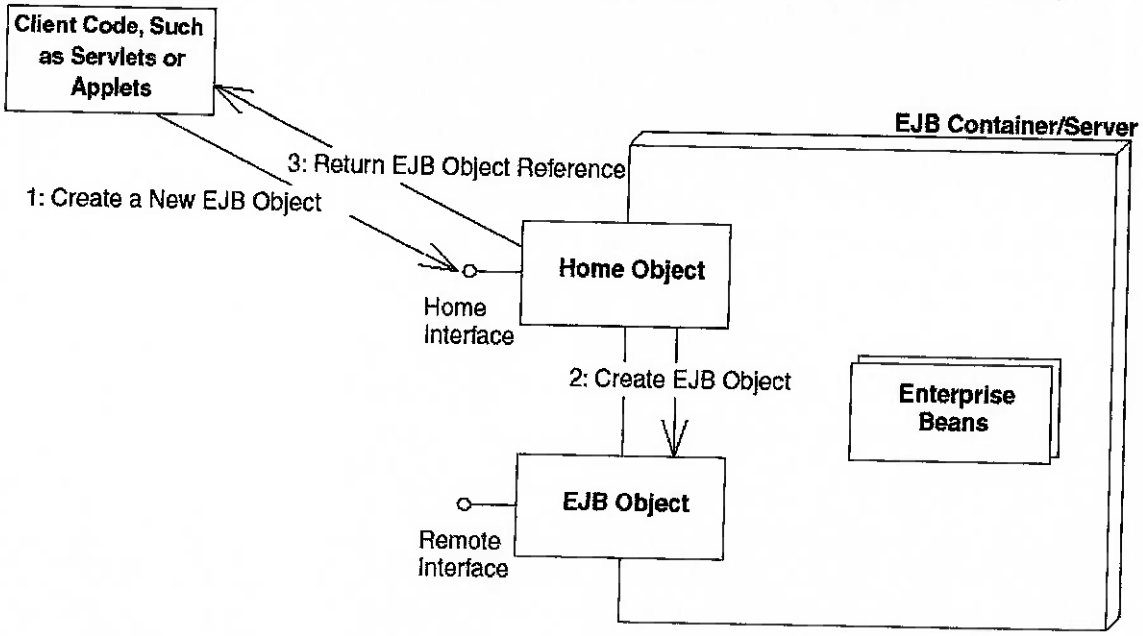


Figura 8. Exemplo de criação de um objeto EJB por meio do objeto home (Roman et.al., 2002).

3.2.3 JSP e *Servlets*

A especificação J2EE inclui também orientações no sentido de como os clientes irão acessar a aplicação. Os principais elementos para isso são páginas JSP e *Servlets* (ver Figura 6).

As primeiras tecnologias para servir aplicações para Internet eram denominada CGI (Common Gateway Interface). No entanto elas tinham a limitação de iniciar um novo processo para cada requisição do cliente, o que prejudicava a aplicação em termos de performance, além de não permitir a manutenção do estado de uma seção entre duas requisições consecutivas (Hanna, 2001).

Vantagens dos Servlets

Os *Servlets* surgiram para superar essas limitações. O servidor de aplicação responsável por instanciar um *servlet* o mantém na memória e utiliza um conjunto disponível de *threads* para atender às requisições. Isso melhora consideravelmente a **performance**. Além disso a API de *servlets* contém uma classe chamada HttpSession que lembra as requisições consecutivas do cliente mantendo ativo o **estado de uma sessão**. Por fim o *servlet* tem as vantagens de **simplicidade**, por ser implementado em um ambiente servidor controlado (ao contrário dos Applets, que são executados no cliente) e do **acesso à tecnologia java** e todas as bibliotecas da especificação (Hanna, 2001).

Vantagens do JSP

As páginas JSP constituem um passo à frente em relação aos Servlets (embora não sejam exatamente um substituto). Elas têm todas essas vantagens descritas para os Servlets além de serem **recompiladas automaticamente** quando necessário, oferecerem um acesso mais simples por existirem no domínio do **servidor Web** e terem maior **compatibilidade com ferramentas de desenvolvimento** diversas por serem parecidas com HTML (Hanna, 2001).

A página JSP é formada por um arquivo que mistura código java e HTML separados por tags especiais. Quando uma página JSP é requisitada, o servidor Web verifica se houve modificação no arquivo. Se houve ele recompila esse arquivo em um *servlet* (que contém o código para o processamento da requisição e para a geração da página HTML) e processa a requisição. Se não houve alteração ele utiliza o servlet já compilado anteriormente para atender à requisição (idem).

Nesse projeto toda a interface do sistema será construída utilizando páginas JSP.

3.3 Modelagem

Booch, Rumbaugh e Jacobson (1995) afirmam que "*Um modelo é uma simplificação da realidade*". Segundo esses autores os modelos são construídos para melhor se entender o sistema desenvolvido e permitem que um problema complexo seja entendido por meio de sua subdivisão em problemas menores, além de permitirem que os seguintes objetivos sejam atingidos:

- visualização do sistema como ele é ou como queríamos que fosse,
- especificação da estrutura ou comportamento do sistema,
- obtenção de um formato para guiar a construção do modelo e
- documentação das decisões feitas.

Ultimamente tem ganhado força e se tornado padrão em desenvolvimento de software a modelagem orientada a objetos. A orientação a objeto surgiu como uma forma de administrar a crescente complexidade no desenvolvimento de software (Booch, 1994).

3.3.1 Elementos da modelagem orientada a objetos

Fazemos aqui um apanhado dos principais conceitos de orientação a objeto de acordo com Booch, 1994; Booch, Rumbaugh e Jacobson (1995); Eckel (2000) entre outros.

Classes

As classes são o elemento básico de um sistema orientado a objetos. Elas representam "coisas" relativas ao domínio do problema. Uma instância de uma classe é um objeto.

Uma classe tem **atributos**. Os atributos definem as características de cada objeto. Além disso as classes têm **métodos**. Um método é uma ação que o objeto dessa classe pode realizar.

Interface

Uma classe define uma interface. Essa interface corresponde aos métodos e atributos que podem ser acessados por um cliente utilizando essa classe.

Encapsulamento

A definição da interface pela classe independe da implementação. Essa é uma das características mais importantes do paradigma de orientação a objeto e sua conseqüência imediata é que o provedor de uma classe pode alterar a implementação sem que o cliente precise ser mudado. Essa característica implica em ganho enorme de flexibilidade e segurança no desenvolvimento, sendo um dos principais avanços com relação ao controle da complexidade.

Reutilização da implementação

O uso de orientação a objeto permite que uma implementação desenvolvida e testada seja utilizada por outras classes. Isso resulta em grande flexibilidade e simplificação no desenvolvimento, já que componentes já desenvolvidos podem ser reutilizados e uma mudança na implementação desses componentes pode ser feita de forma transparente sem que todo o sistema precise ser alterado.

Essa reutilização é denominada **composição** (pois uma classe passa a ser composta por outras) ou **agregação** (quando uma classe agrega outras).

Reutilização da interface

Da mesma forma, a modelagem orientada a objetos permite que seja criada uma classe que use a mesma interface de outra já existente e estenda essa interface. Essa figura é denominada **herança**, pois a nova classe herda as propriedades de outra classe. Além disso essa herança pode envolver a mudança de partes da interface, o que é denominado **polimorfismo**.

3.3.2 UML

A UML – Unified Modeling Language (Linguagem unificada de modelagem) surgiu com o intuito de padronizar a modelagem orientada a objetos.

Trata-se de uma linguagem gráfica para visualizar, especificar, construir e documentar os diversos elementos que compõe um sistema orientado a objetos (Booch, Rumbaugh e Jacobson; 1995).

A linguagem UML é extensa e envolve uma série de componentes que estão divididos em quatro tipos:

- **Estruturais**: são as partes estáticas do modelo (classes, interfaces, colaborações, use cases, componentes, nós)
- **De comportamento**: partes dinâmicas (mensagens e estados)
- **De agrupamento**: os *packages*.
- **De anotação**: caixas de texto explicativas.

Esses componentes podem ser dispostos em uma série de diagramas com o objetivo de representar diversas características do sistema. Os principais diagramas que serão utilizados no projeto estão explicados a seguir.

- **Use Cases**: os use cases representam os requisitos de funcionamento e a forma como o usuário irá interagir com o sistema. O diagrama de use case mostra os requisitos e a forma como o usuário interage com o mesmo.

- **Diagramas de Classes**: os diagramas de classe representam as classes do sistema e suas relações.
- **Diagramas de Seqüência**: os diagramas de seqüência representam como as classes interagem para realizar algum requisito. Normalmente existe um diagrama de seqüência para cada *use case*.

Exemplos desses diagramas serão mostrados no capítulo 4 que discute a modelagem do sistema.

Existem vários outros diagramas especificados em UML. No entanto nesse projeto nós iremos utilizar somente esses apresentados conforme será discutido no próximo item.

3.3.3 Abordagem para o projeto

Existem diversas abordagens para a modelagem e construção de um sistema. Entre elas podemos destacar:

- **Guiada pelos requisitos (*use case driven*)**
- **Foco na arquitetura (*architecture-centric*)**
- **Iterativa e incremental**

Nesse projeto iremos adotar a abordagem guiada pelos requisitos. Isso implica que a primeira consideração será o conjunto de requisitos do sistema e a forma como o usuário (ou outros sistemas) irá interagir com os mesmos. Eventualmente a definição dos requisitos envolve também a definição de interfaces para o sistema (como formulários ou GUIs).

De acordo com essa abordagem, depois que todos os requisitos forem identificados, nós partiremos para a identificação das classes que compõe o domínio do problema.

Tendo os *use cases* e as classes, partiremos para a definição da forma como cada classe irá interagir no sistema para realizar os *use cases* propostos. Isso será feito desenhando um diagrama de seqüência para cada *use case*.

Tendo os diagramas de seqüência e de classes iremos então gerar o código e implementar o sistema em um ambiente de teste.

É claro que essa abordagem não elimina a necessidade de transitar entre as fases de forma iterativa para corrigir continuamente os eventuais problemas na definição do modelo.

No capítulo 4 será apresentado todo o desenvolvimento do modelo para nosso projeto de acordo com essa abordagem aqui discutida.

4 Especificação do Projeto

4.1 Introdução

Nessa seção será apresentada com detalhes e comentários toda a documentação do projeto do sistema conforme discutido no item 3.3.

O item 4.2 apresenta todos os *use cases* considerados no levantamento dos requisitos. A identificação dos mesmo, como discutido, é o ponto de partida para o projeto do sistema. O item 4.3 apresenta as classes identificadas. Finalmente no item 4.4 são apresentados os diagramas de seqüência que procuram representar de forma simples como as classes irão interagir para executar os requisitos especificados. Posteriormente são apresentados, nos itens 4.5 e 4.6 os requisitos não funcionais e a perspectiva gerencial do projeto.

4.2 Use Cases

Primeiramente os *use cases* foram agrupados de acordo com os *packages* mostrados na Figura 9.

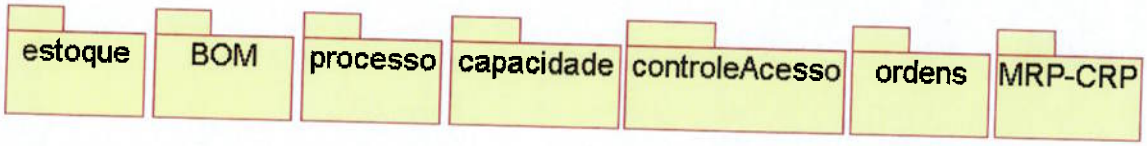


Figura 9. Packages para os use cases.

A seguir são descritos cada *use case* agrupando-se os mesmo pelos *packages*.

4.2.1 Estoque

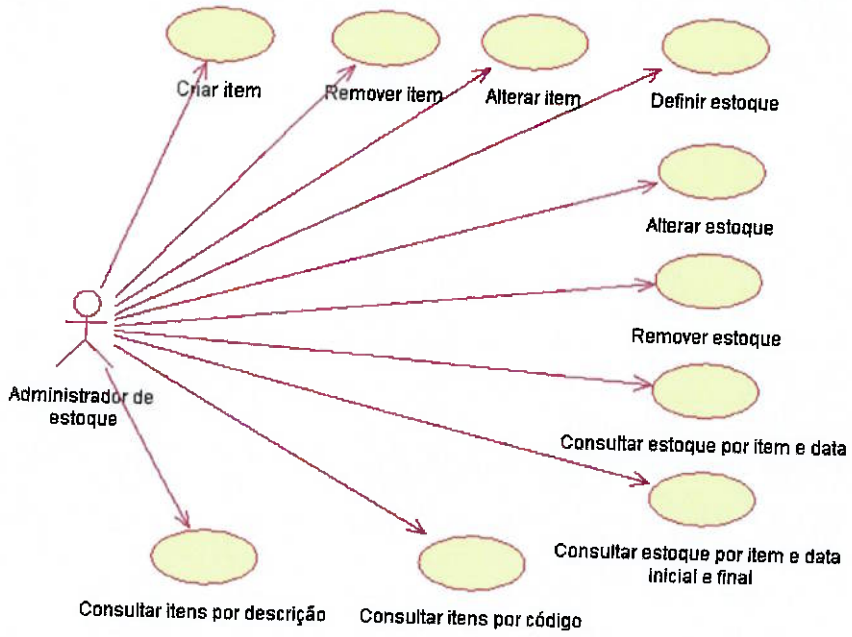


Figura 10. *Package* Estoque com os *use cases*.

Nesse *package* foram incluídas todas as ações referentes ao tratamento da lista mestre de materiais e do estoque. Isso incluí criação, alteração e remoção de itens e de registros de estoque para um item em uma data, além de diversas consultas. Seguem as descrições de cada *use case*. Eventualmente são apresentados também exemplos de interfaces utilizadas pelo sistema.

Criar item

1. O operador entra o código, a descrição, seleções (produzido ou comprado, real ou fantasma, final ou intermediário), seleção que diz se o item entra ou não no cálculo de MRP, a unidade de medida, o custo, tamanho do Lote, *lead time* (tempo de compra/produção) e o estoque de segurança.

2. O LLC - *low level code* (nível mais baixo na estrutura para este item) é calculado pelo sistema (campo desabilitado).

3. Para o campo Estoque de segurança, se não é fornecido um valor o sistema assume o valor zero.

4. Para o campo Cálculo por MRP, se não é fornecido um valor o sistema assume que seja Sim.

5. Todos os outros campos são obrigatórios para a inserção do item.

6. Se algum campo obrigatório não é fornecido o formulário não é submetido (verificado em script no cliente). Se algum tipo de dado estiver errado ou o código estiver duplicado o sistema retorna o erro, recompõe a tela com os dados fornecidos anteriormente a não ser pelo dado errado, e solicita que o usuário complete as informações erradas.

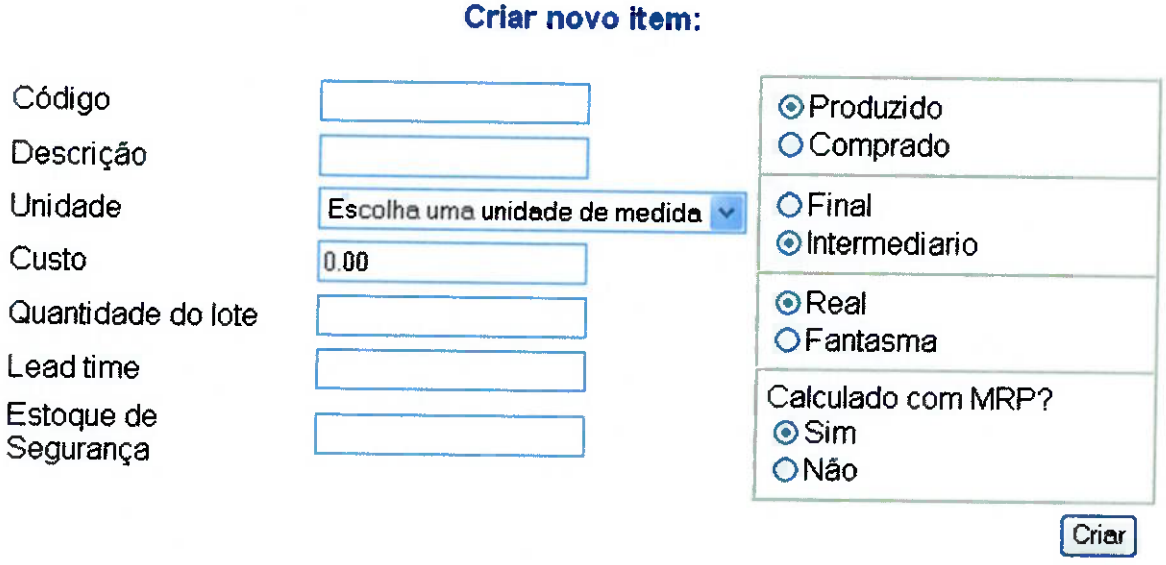


Figura 11. Exemplo de interface para criação de itens

Alterar item

1. Para alterar um item o operador escolhe o item previamente por meio de um mecanismo de busca. Ele dispõe então de um formulário igual ao de inserção.

2. O operador altera a descrição, seleções (produzido ou comprado, real ou fantasma, final ou intermediário), seleção que diz se o item entra ou não no cálculo de MRP, a unidade de medida e o estoque de segurança, conforme necessário.

4. Para o campo Estoque de segurança, se não é fornecido um valor o sistema assume o valor zero.

5. Para o campo Cálculo por MRP, se não é fornecido um valor o sistema assume que seja Sim.

6. Todos os outros campos são obrigatórios para a inserção do item.

7. Se algum campo obrigatório não é fornecido o formulário não é submetido (verificado em script no cliente). Se algum tipo de dado estiver errado ou o código estiver duplicado o sistema retorna o erro, recompõe a tela com os dados fornecidos anteriormente a não ser pelo dado errado, e solicita que o usuário complete as informações erradas.

Remover item

1. Para remover um item o operador escolhe o item previamente por meio de um mecanismo de busca. O resultado da busca permite que o operador remova os itens recuperados.

2. O sistema ainda pede uma confirmação ao usuário antes de realizar a operação, avisando que todos os itens da BOM e do Estoque referentes a esse item serão também removidos.

Consultar itens por Descrição

1. O operador fornece a descrição ou parte dela referente a um item.

2. O sistema monta uma ou mais telas de resultado com os itens cuja descrição contém a "string" fornecida na busca e suas características.

3. Os itens serão mostrados de 10 em 10 com a descrição em ordem alfabética

4. Para cada item retornado o sistema irá disponibilizar as seguintes operações.

· Alterar / Remover item

· Pesquisar estrutura de produtos abaixo deste item

· Pesquisa "*where-used*" do item

· Pesquisar estoque para este item

· Pesquisar demanda para este item

Consultar itens por Código

1. O operador fornece o código do item diretamente e o sistema retorna uma tela com um único item e suas características.

2. Para o item retornado o sistema irá disponibilizar as seguintes operações.

· Alterar / Remover item

· Pesquisar estrutura de produtos abaixo deste item

· Pesquisa "where-used" do item

· Pesquisar estoque para este item

· Pesquisar demanda para este item

Definir estoque

1. O operador fornece ao sistema o código do item, a data para definição do estoque, as necessidades brutas dependente e independente, o recebimento programado e, eventualmente, o estoque projetado nessa data.

2. Caso não saiba o código do item ele tem a possibilidade de realizar uma busca.

3. Se o código ou a data forem inválidos o sistema retorna uma mensagem de inconsistência e volta para a tela anterior com os dados enviados.

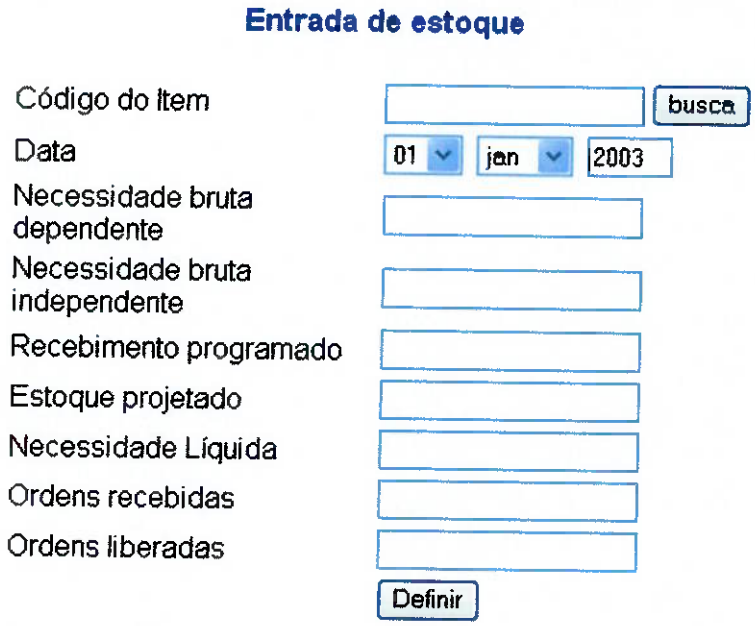


Figura 12. Exemplo de interface para definição de estoque.

Alterar estoque

1. Para alterar o estoque em uma data o operador fornece inicialmente ao sistema o código do item e a data para definição do estoque.

2. O sistema monta então uma tela semelhante à da inserção para que o operador altere as necessidades brutas dependente e independente, o recebimento programado e o estoque final.

3. Caso não se saiba o código do item há a possibilidade de realizar uma busca nos itens.

Remover estoque

1. Para remover do estoque um item em uma data o operador fornece inicialmente ao sistema o código do item e a data.

2. O sistema retorna então o resultado da busca por esse registro com a opção de remover.

3. O sistema ainda pede uma confirmação ao usuário antes de realizar a operação.

4. Caso não se saiba o código do item há a possibilidade de realizar uma busca nos itens.

Consultar estoque por Item e Data

1. Para realizar a busca o operador fornece ao sistema o código do item e a data.

2. Caso não se saiba o código do item há a possibilidade de realizar uma busca nos itens.

Consultar estoque por Item e Data inicial e final

1. Para essa consulta o operador fornece o código do item e as datas inicial e final.

2. Caso não se saiba o código do item há a possibilidade de realizar uma busca nos itens.

3. O sistema irá montar uma tela com uma tabela horizontal do estoque dia a dia.

Lapiseira | LT: 1.0 | ES: 1000.0 | Lote: 1.0	2003-06-29	2003-06-30	2003-07-01	2003-07-02	2003-07-03	2003-07-04	2003-07-05	2003-07-06	2003-07-07
Nec. Bruta Dep.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Nec. Bruta Indep	0.0	0.0	5000.0	5000.0	5000.0	2500.0	10000.0	10000.0	6000.0
Receb. Programado	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Estoque Projetado	0.0	1000.0	1000.0	1000.0	1000.0	1000.0	1000.0	1000.0	1000.0
Necessidade Líquida	0.0	800.0	5000.0	5000.0	5000.0	2500.0	10000.0	10000.0	6000.0
Ordens Recebidas	0.0	800.0	5000.0	5000.0	5000.0	2500.0	10000.0	10000.0	6000.0
Ordens Liberadas	800.0	5000.0	5000.0	5000.0	2500.0	10000.0	10000.0	6000.0	0.0

Figura 13. Exemplo de resultado de busca na manipulação de estoque

4.2.2 BOM – Estrutura de materiais

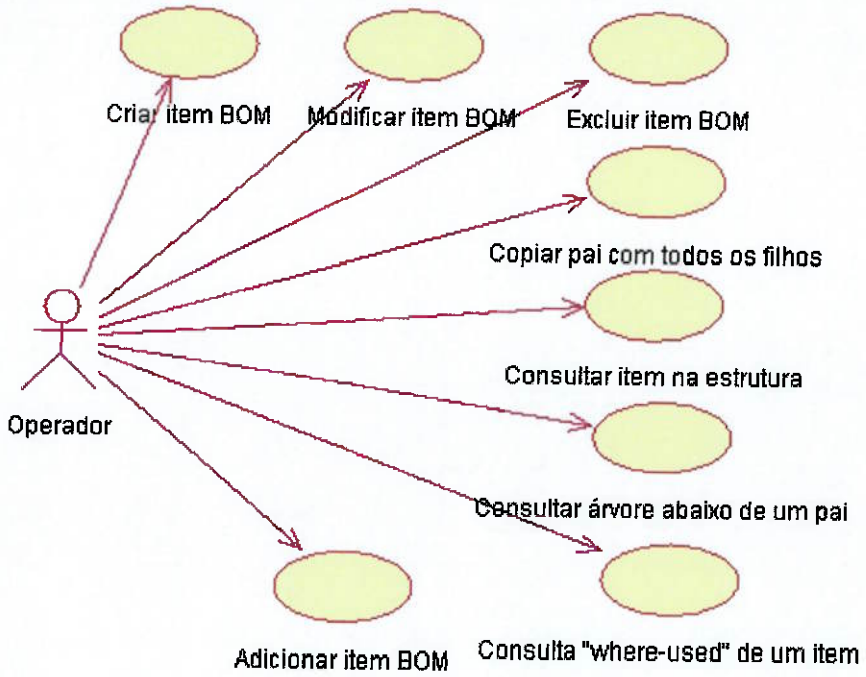


Figura 14. *Package* BOM com todos os *use cases*.

Nesse *package* são tratadas todas as ações de manipulação das listas de materiais (BOM – *Bill of Materials*). Isso inclui as operações de inclusão, alteração, remoção e consulta das árvores de materiais.

Criar nova árvore

1. O operador define o código da estrutura (considerando-se que o mesmo item pai pode ter mais de uma estrutura), o código do item e uma descrição para a estrutura.

2. Na entrada dos dados o sistema verifica se o código do item é válido, se este é um "item final" (de demanda independente) e se ainda não existe o mesmo código para outra estrutura

3. Se estiver correto, o item é encontrado, todas as variáveis são setadas e é criado um item BOM com o atributo pai nulo.

4. Se não for um "item final", ou já existir o código para a estrutura o sistema retorna uma mensagem de erro e recompõe a tela para que o operador faça as correções.

Modificar item BOM

1. O operador fornece o código do item BOM.

2. O sistema monta um resultado com as características do item nessa estrutura e a informação se este item tem filhos, permitindo a alteração dos atributos descrição e qtde.

3. Se não for encontrado o item ou não for encontrada a estrutura o sistema informa o erro e recompõe a tela.

4. O atributo pai não pode ser alterado. Ele só é modificado quando o item é criado ou quando é adicionado ou removido de um Pai.

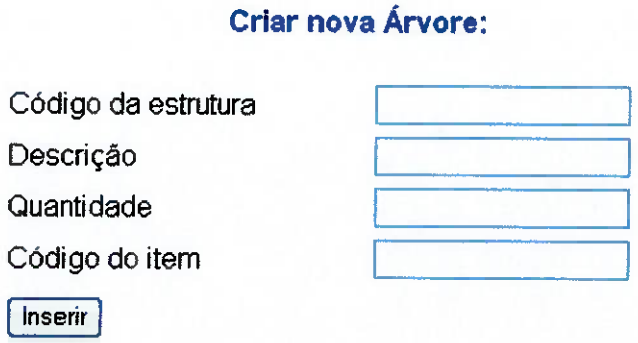


Figura 15. Exemplo de interface para a criação de item Pai (nova árvore).

Adicionar item BOM

1. O operador entra o código do item BOM (ao qual será adicionado o item) e o código do item a ser adicionado.

3. O operador tem a possibilidade de realizar uma busca para escolha do item.

4. O sistema irá então validar o código fornecido, criar um item Filho com o atributo pai, encontrar o item correspondente e setar as variáveis no novo itemBOM criado, além de adicionar o novo Filho criado na tabela hash do Pai.

5. Se não for encontrado o item nessa estrutura ou não for encontrada a estrutura o sistema informa o erro e recompõe a tela.

Excluir item BOM

1. O operador entra o código do item BOM a ser excluído.

2. Se o item for encontrado, o sistema verifica se tem filhos.

3. Se tem o sistema pede uma confirmação da exclusão deste item e de toda a árvore abaixo dele. Se não pede uma confirmação da exclusão do item.

4. Se a exclusão é confirmada, o sistema percorre toda a lista de filhos seguindo os passos:

4.1. Se o item não tem filhos ele é removido da tabela Hash do pai e apagado.

4.2. Se o item tem filhos, todos os filhos são apagados pela mesma rotina (recursivamente) e depois ele é removido da tabela hash do pai e apagado.

Copiar pai com todos os filhos

1. O operador entra o código do item BOM e o código para a nova estrutura.

2. Se o sistema não encontra o item retorna um erro e recompõe a tela.

3. Se o item é encontrado é criado um novo item pai com o novo código para a estrutura e todos os filhos são adicionados recursivamente até que seja percorrida toda a estrutura.

Consultar item na estrutura

1. O operador fornece o código do item e o sistema retorna o item com suas características.

2. O operador tem a possibilidade de realizar uma consulta da árvore ou *where-used*, ou modificar ou alterar o item.

Consultar árvore abaixo de um pai

1. O operador entra o código do item BOM e o código da estrutura.

2. Se o sistema não encontra o item como pai nessa estrutura ele retorna um erro e recompoe a tela.

3. Se o item é encontrado a estrutura é percorrida a partir do item pai e a árvore é exibida.

Consulta *where-used* de um item

1. O operador entra o código do item.

2. O sistema retorna todas as ocorrências desse item em todas as estruturas com os respectivos pais.

4.2.3 Processos

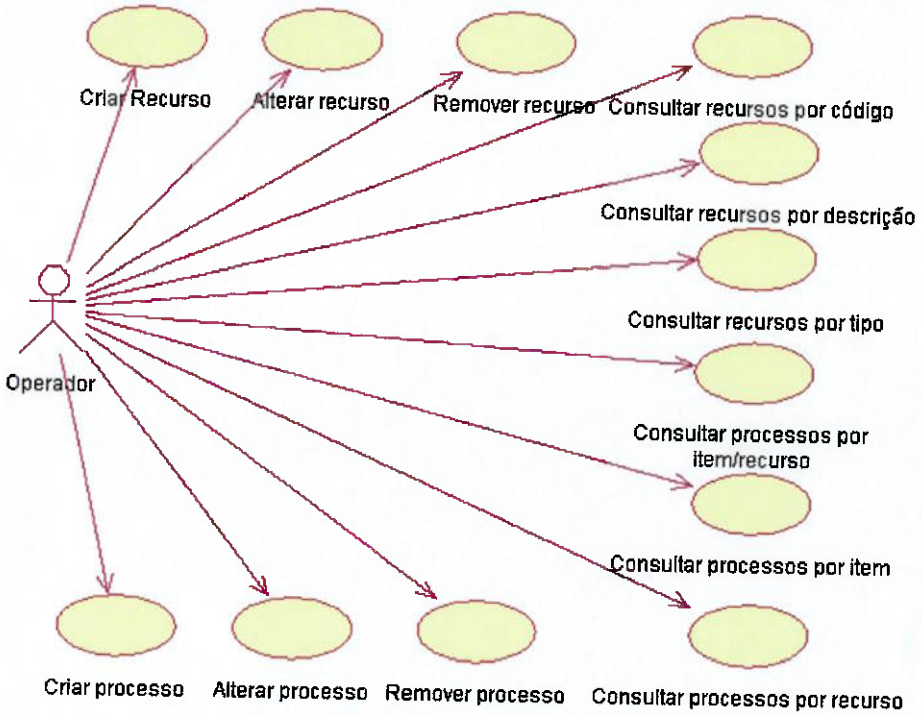


Figura 16. *Package* processo com todos os *use cases*.

Esse *package* envolve todas as operações referentes aos recursos e processos, incluindo operações de criar, remover, alterar e diversas consultas.

Criar recurso

1. O operador entra o código do recurso, a descrição, o tipo, o tempo de setup e disponibilidade de horas de uma unidade do recurso.

2. O sistema cria um novo recurso com a informação fornecida e mostra o resultado.

3. Caso o código já exista o sistema retorna um erro e recompoe a tela.

4. Todos os campos são obrigatórios. Para o setup se não é fornecido um valor o sistema assume zero.

Alterar recurso

1. O operador entra o código do recurso a ser modificado.

2. Caso não seja encontrado o recurso, o sistema retorna um erro.

3. Do contrário o sistema monta uma interface para alteração da descrição, tipo, tempo de setup e disponibilidade de horas.

4. Todos os campos são obrigatórios. Para o setup se não é fornecido um valor o sistema assume zero.

Remover recurso

1. O operador entra o código do recurso a ser removido.

2. Caso não seja encontrado o recurso, o sistema retorna um erro.

3. Do contrário o sistema retorna o recurso com suas características e pede confirmação antes de remover.

4. Se existirem processos e capacidade definida relativos ao recurso o sistema confirma a deleção de todas as instâncias.

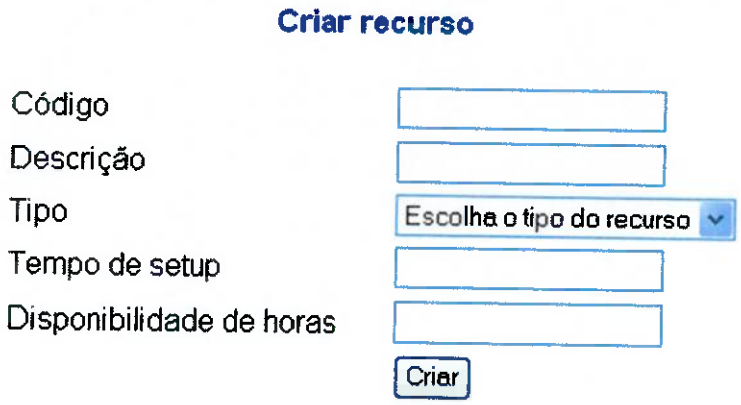


Figura 17. Exemplo de interface para criação de recurso

Consultar recursos por código

1. O operador entra o código do recurso e este é recuperado pelo sistema.

2. Com o recurso recuperado o operador tem a possibilidade de realizar as seguintes operações:

- Alteração/remoção do recurso
- Consulta de processos para o recurso
- Consulta de capacidade para o recurso

Consultar recursos por descrição

1. O operador entra a descrição ou parte dela. O sistema recupera todos os recursos cujas descrições contenham a *string* fornecida.

2. Os recursos serão exibidos em ordem alfabética de dez em dez.

3. Para cada recurso recuperado o operador tem a possibilidade de realizar as seguintes operações:

- Alteração/remoção do recurso
- Consulta de processos para o recurso

- Consulta de capacidade para o recurso

Consultar recursos por tipo

1. O operador entra o tipo do recurso e o sistema recupera todos os recursos desse tipo.

2. Os recursos serão exibidos em ordem alfabética de dez em dez.

3. Para cada recurso recuperado o operador tem a possibilidade de realizar as seguintes operações:

- Alteração/remoção do recurso
- Consulta de processos para o recurso
- Consulta de capacidade para o recurso

Criar processo

1. O operador entra o código do item BOM, o código do recurso, a descrição, a produtividade e o tempo de setup do processo.

2. O sistema cria um novo processo com a informação fornecida e mostra o resultado.

3. Caso já exista um processo para este item BOM e este recurso o sistema pergunta se o operador quer substituir o processo.

4. Todos os campos são obrigatórios. Para o setup se não é fornecido um valor o sistema assume o setup do recurso.

5. O operador tem a possibilidade de realizar uma busca na estrutura e nos recursos para definir o código.

Alterar processo

1. O operador entra o código do item BOM e o código do recurso para o processo a ser alterado.

2. Caso não exista um processo para este item BOM e este recurso o sistema retorna um erro.

3. Do contrário o sistema monta um resultado com o processo especificado permitindo a edição da descrição, setup e produtividade.

4. Todos os campos são obrigatórios. Para o setup se não é fornecido um valor o sistema assume o setup do recurso.

5. O operador tem a possibilidade de realizar uma busca na estrutura e nos recursos para definir o código.

Remover processo

1. O operador entra o código do item BOM e o código do recurso para o processo a ser removido.

2. Caso não exista um processo para este item BOM e este recurso o sistema retorna um erro.

3. Do contrário o sistema pede uma confirmação antes de remover o processo.

Consultar processos por Item/Recurso

1. O operador entra o código do item BOM e o código do recurso para o processo.

2. Caso não exista um processo para este item BOM e este recurso o sistema retorna um erro.

3. Do contrário o sistema monta um resultado com o processo.

Consultar processos por Item

1. O operador entra o código do item BOM.

2. Caso não exista um processo para este item BOM sistema retorna um erro.

3. Do contrário o sistema monta um resultado com os processos encontrados.

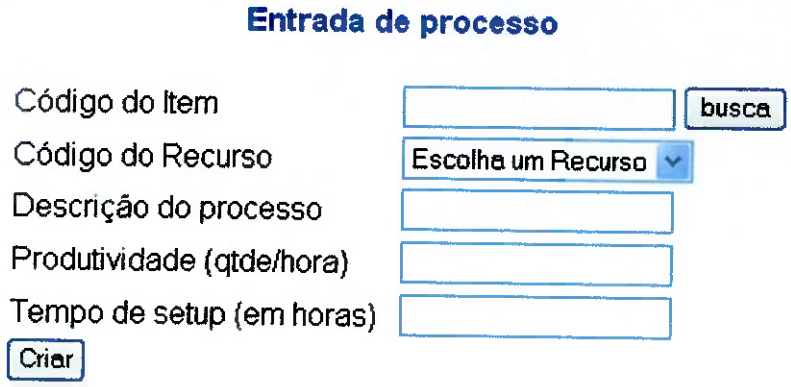


Figura 18. Exemplo de interface para a criação de processo

Consultar processos por Recurso

1. O operador entra o código do recurso.

2. Caso não exista um processo para este recurso sistema retorna um erro.

3. Do contrário o sistema monta um resultado com os processos encontrados.

4. Os processos serão exibidos de dez em dez ordenados por item BOM.

4.2.4 Capacidade

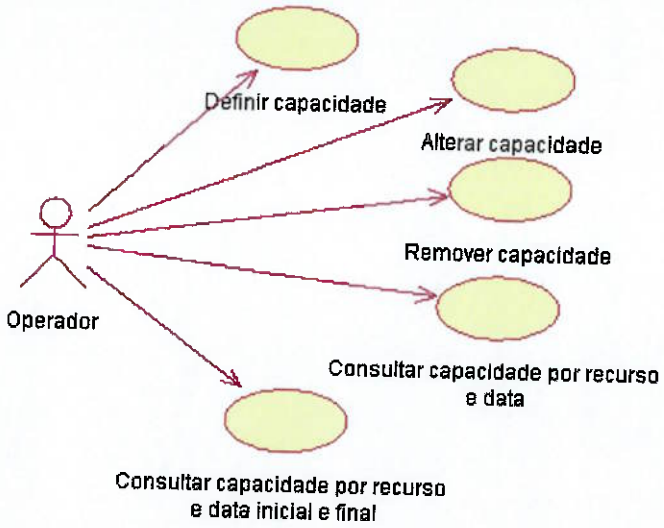


Figura 19. *Package* capacidade com todos os *use cases*.

Esse *package* trata exclusivamente da definição e manipulação em geral da capacidade, ou seja, do nível de utilização dos recursos período a período.

Definir capacidade

1. O operador fornece ao sistema o código do recurso, a data, e a quantidade disponível do recurso.

2. Caso não saiba o código do recurso ele tem a possibilidade de realizar uma busca.

3. Se o código ou a data forem inválidos o sistema retorna uma mensagem de inconsistência e volta para a tela anterior com os dados enviados.

4. Todos os outros dados referentes à capacidade nessa data são calculados pelo sistema e aparecem no formulário como campos desabilitados.

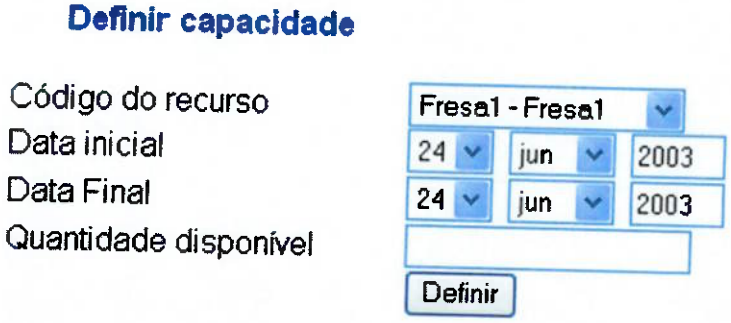


Figura 20. Exemplo de interface para definição de capacidade.

Alterar capacidade

1. O operador fornece ao sistema o código do recurso e a data.

2. Caso não saiba o código do recurso ele tem a possibilidade de realizar uma busca.

3. Se o código ou a data forem inválidos o sistema retorna uma mensagem de inconsistência.

4. Do contrário monta um formulário com o resultado da busca permitindo a edição do campo quantidade disponível.

Remover capacidade

1. O operador fornece ao sistema o código do recurso e a data.

2. Caso não saiba o código do recurso ele tem a possibilidade de realizar uma busca.

3. O sistema ainda pede uma confirmação antes de excluir a capacidade

Consultar capacidade por recurso e data

1. O operador entra o código do recurso e a data.

2. Caso não saiba o código do recurso ele tem a possibilidade de realizar uma busca.

Consultar capacidade por recurso e data inicial e final

1. O operador entra o código do recurso e as datas inicial e final.

2. Caso não saiba o código do recurso ele tem a possibilidade de realizar uma busca.

3. O sistema monta o resultado na forma de uma tabela horizontal mostrando a capacidade dia a dia.

4.2.5 Ordens

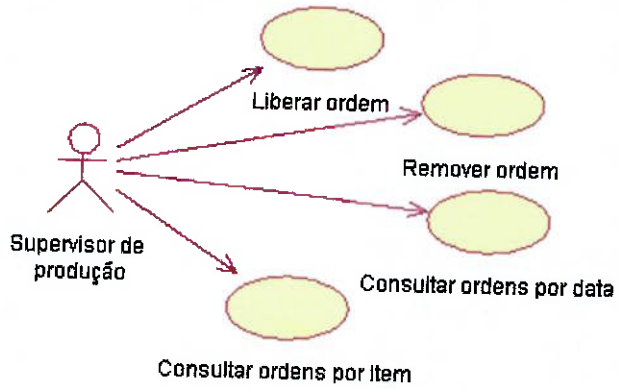


Figura 21. *Package* ordens com todos os *use cases*.

Esse *package* lida com as ordens geradas pelo MRP. Ele compreende as operações de liberar ou remover as ordens e consultá-las por item ou por data. Nota-se que uma ordem não pode ser criada pois estamos assumindo que todas as ordens aqui tratadas são geradas pelo MRP. É claro que irão existir ordens de compra e produção não geradas pelo

MRP mas estamos assumindo que isso está em um módulo externo não compreendido nesse projeto.

Liberar ordem

1. O supervisor realiza uma busca que traz as ordens referentes à busca com a possibilidade de liberar cada uma.

2. Se a ordem for liberada o sistema promove a alteração do recebimento programado na tabela de estoque.

Remover ordem

1. O supervisor realiza uma busca que traz as ordens referentes à busca com a possibilidade de remover cada uma.

2. O sistema ainda pede uma confirmação antes de remover a ordem.

Consultar ordens por data

1. O supervisor entra a data e o sistema monta o resultado com todas as ordens colocadas nessa data.

Consultar ordens por item

1. O supervisor entra o código do item e o sistema monta o resultado com todas as ordens colocadas para esse item.

2. Ele ainda tem a possibilidade de realizar uma busca para encontrar o item.

4.2.6 Controle de acesso

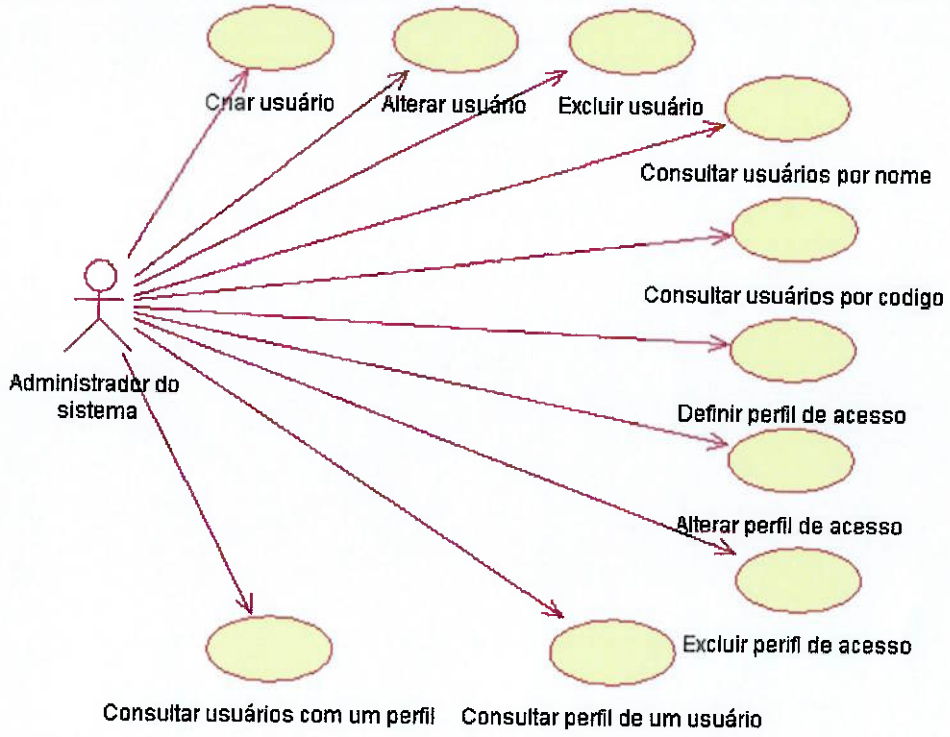


Figura 22. *Package* controle de acesso com todos os *use cases*.

4.2.7 MRP / CRP

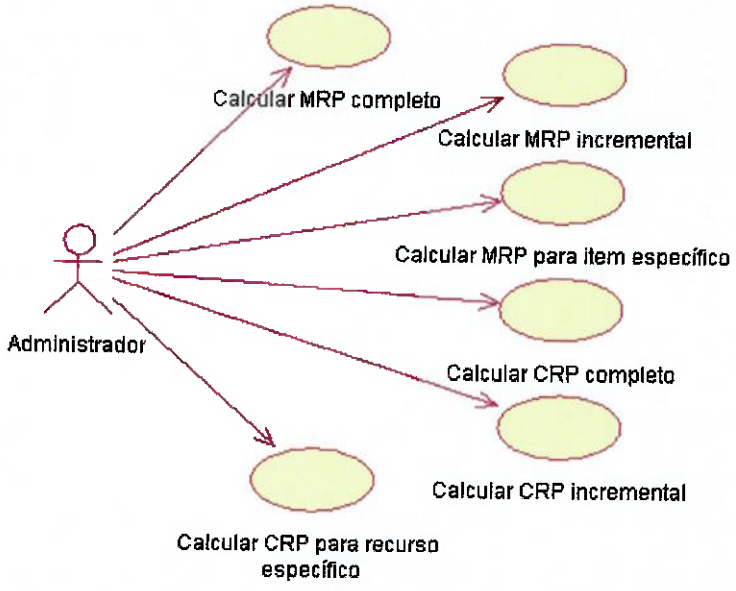


Figura 23. *Package* MRP / CRP com todos os *use cases*.

Esse *package* compreende as operações de administração do sistema para disparar os processamentos do MRP e do CRP conforme explicado nos itens 2.2 e 2.3. Não iremos discutir os *use cases* por se tratarem apenas de ações pontuais.

4.3 Modelo de Classes

A análise dos requisitos e o entendimento da aplicação, conforme explicada com detalhes no item 2 fornece os meios para se identificar as classes que farão parte do nosso sistema. Essas classes estão dispostas no diagrama na Figura 24.

O diagrama está feito segundo o padrão UML e é auto-explicativo, mostrando inclusive os tipos de relacionamento entre as classes. Os atributos também já foram discutidos no item 2, a não ser por alguns detalhes. Os métodos expostos no diagrama são os obrigatórios para os EJB (gerador pelo Rational Rose®). Os métodos de negócio serão obtidos durante nosso próximo passo no projeto.

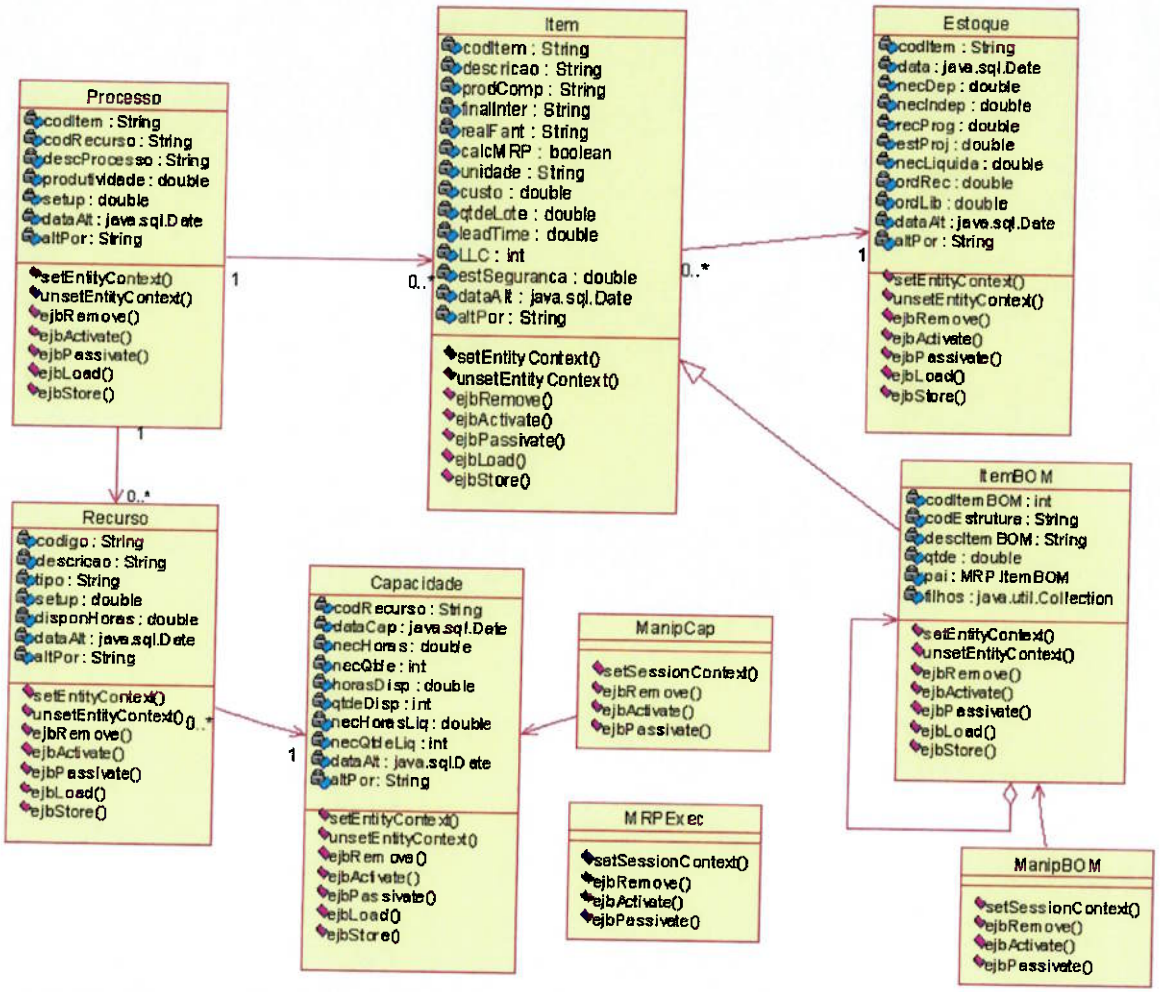


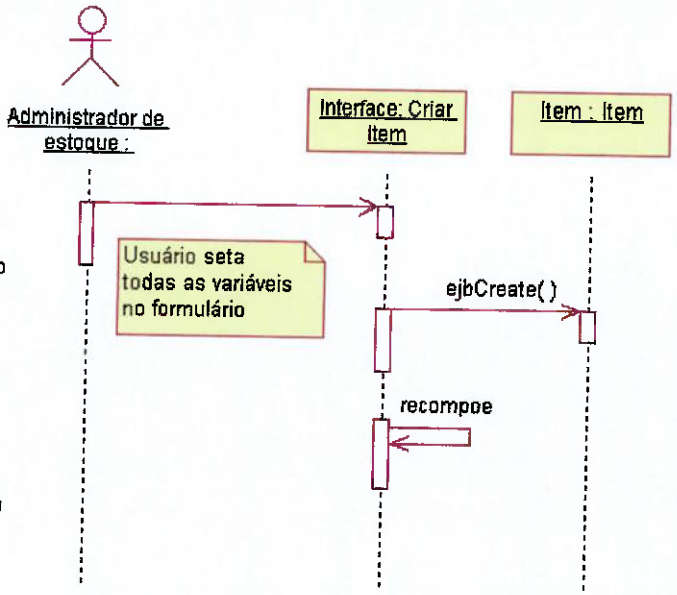
Figura 24. Diagrama de classes do sistema.

4.4 Diagramas de seqüência

Neste item estão apresentados os diagramas de seqüência para cada *use case* apresentado no item 4.2. Optamos por apresentar cada diagrama de seqüência juntamente com o *use case* correspondente de modo a facilitar a análise.

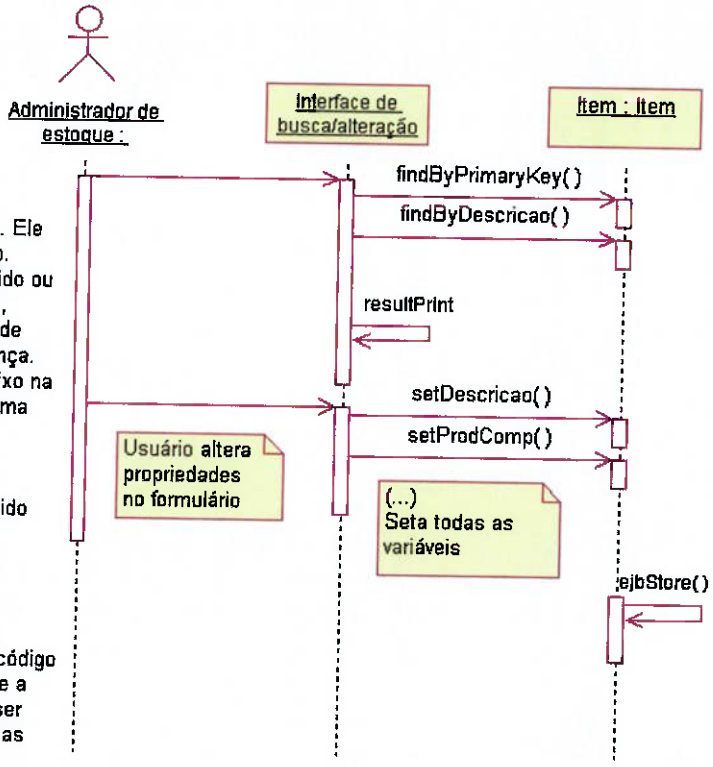
4.4.1 Estoque

Criar item



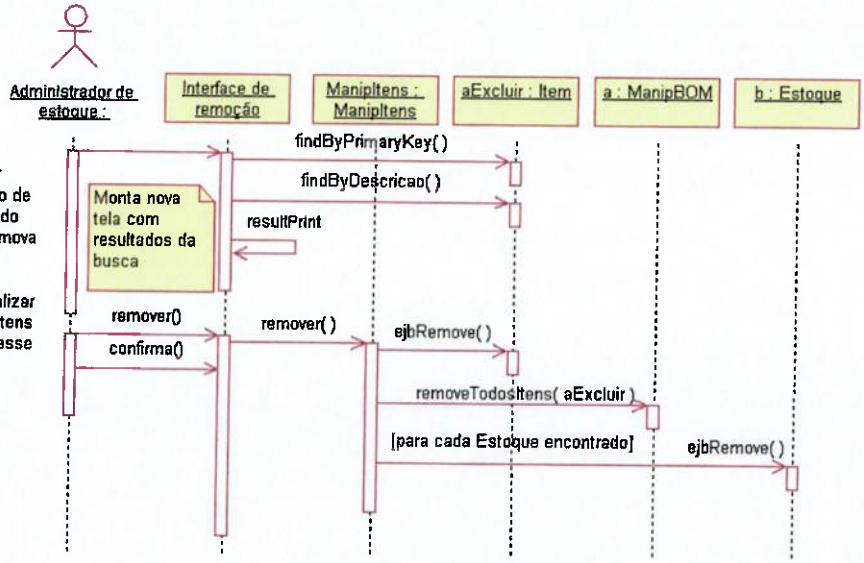
1. O operador entra a descrição, seleções (produzido ou comprado, real ou fantasma, final ou intermediário), seleção que diz se o item entra ou não no cálculo de MRP, a unidade de medida, o custo, tamanho do Lote, lead time (tempo de compra/produção) e o estoque de segurança.
2. O código e o LLC - low level code (nível mais baixo na estrutura para este item) são calculados pelo sistema (campos desabilitados).
3. Para o campo Estoque de segurança, se não é fornecido um valor o sistema assume o valor zero.
4. Para o campo Cálculo por MRP, se não é fornecido um valor o sistema assume que seja Sim.
5. Todos os outros campos são obrigatórios para a inserção do item.
6. Se algum campo obrigatório não é fornecido o formulário não é submetido (verificado em script no cliente). Se algum tipo de dado estiver errado ou o código estiver duplicado o sistema retorna o erro, recompõe a tela com os dados fornecidos anteriormente a não ser pelo dado errado, e solicita que o usuário complete as informações erradas.

Alterar item



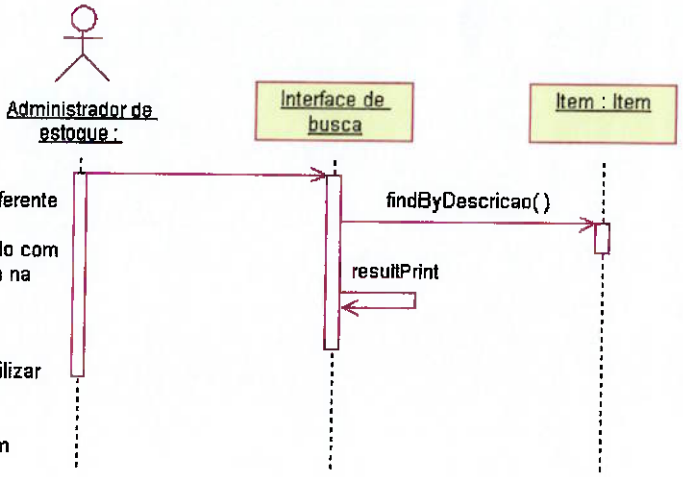
1. Para alterar um item o operador escolhe o item previamente por meio de um mecanismo de busca. Ele dispõe então de um formulário igual ao de inserção.
2. O operador entra a descrição, seleções (produzido ou comprado, real ou fantasma, final ou intermediário), seleção que diz se o item entra ou não no cálculo de MRP, a unidade de medida e o estoque de segurança.
3. O código e o LLC - low level code (nível mais baixo na estrutura para este item) são calculados pelo sistema (campos desabilitados).
4. Para o campo Estoque de segurança, se não é fornecido um valor o sistema assume o valor zero.
5. Para o campo Cálculo por MRP, se não é fornecido um valor o sistema assume que seja Sim.
6. Todos os outros campos são obrigatórios para a inserção do item.
7. Se algum campo obrigatório não é fornecido o formulário não é submetido (verificado em script no cliente). Se algum tipo de dado estiver errado ou o código estiver duplicado o sistema retorna o erro, recompõe a tela com os dados fornecidos anteriormente a não ser pelo dado errado, e solicita que o usuário complete as informações erradas.

Remover item



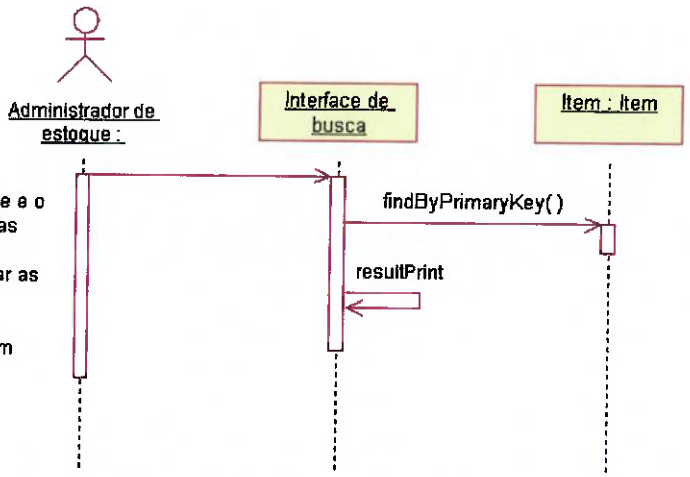
1. Para remover um item o operador escolhe o item previamente por meio de um mecanismo de busca. O resultado da busca permite que o operador remova os itens recuperados.
2. O sistema ainda pede uma confirmação ao usuário antes de realizar a operação, avisando que todos os itens da BOM e do Estoque referentes a esse item serão também removidos.

Consultar itens por Descrição



1. O operador fornece a descrição ou parte dela referente a um item.
2. O sistema monta uma ou mais telas de resultado com os itens cuja descrição contém a "string" fornecida na busca e suas características.
3. Os itens serão mostrados de 10 em 10 com a descrição em ordem alfabética
4. Para cada item retornado o sistema irá disponibilizar as seguintes operações.
· Alterar / Remover item
· Pesquisar estrutura de produtos abaixo deste item
· Pesquisa "where-used" do item
· Pesquisar estoque para este item
· Pesquisar demanda para este item

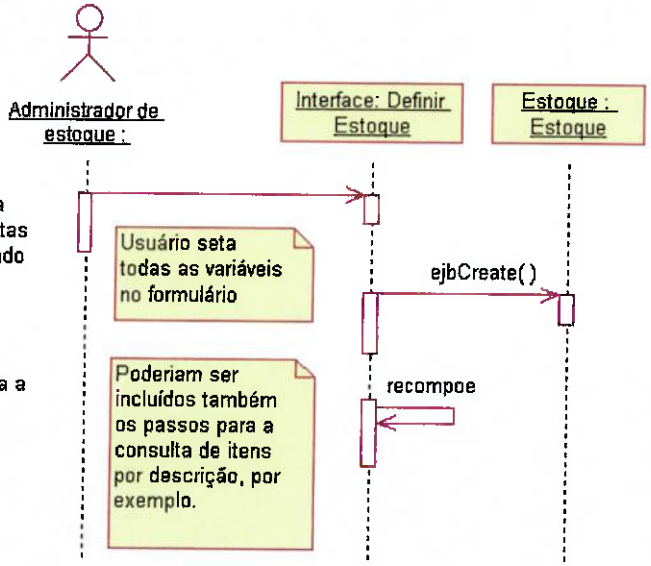
Consultar itens por Código



1. O operador fornece o código do item diretamente e o sistema retorna uma tela com um único item e suas características.
2. Para o item retornado o sistema irá disponibilizar as seguintes operações.
· Alterar / Remover item
· Pesquisar estrutura de produtos abaixo deste item (inserir, alterar ou apagar linha)
· Pesquisa "where-used" do item
· Pesquisar estoque para este item
· Pesquisar demanda para este item

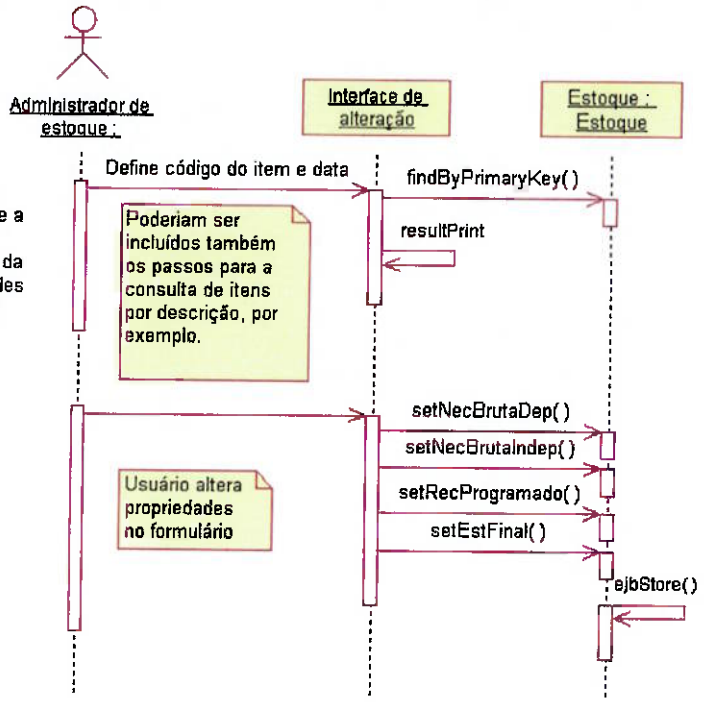
Definir estoque

1. O operador fornece ao sistema o código do item, a data para definição do estoque, as necessidades brutas dependente e independente, o recebimento programado e o estoque final nessa data.
2. Caso não saiba o código do item ele tem a possibilidade de realizar uma busca.
3. Se o código ou a data forem inválidos o sistema retorna uma mensagem de inconsistência e volta para a tela anterior com os dados enviados.

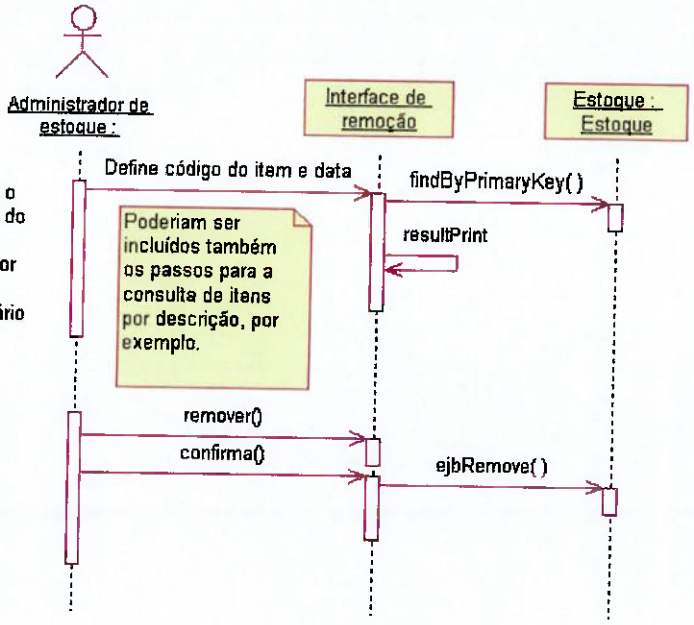


Alterar estoque

1. Para alterar o estoque em uma data o operador fornece inicialmente ao sistema o código do item e a data para definição do estoque.
2. O sistema monta então uma tela semelhante à da inserção para que o operador altere as necessidades brutas dependente e independente, o recebimento programado e o estoque final.
3. Caso não se saiba o código do item há a possibilidade de realizar uma busca nos itens.

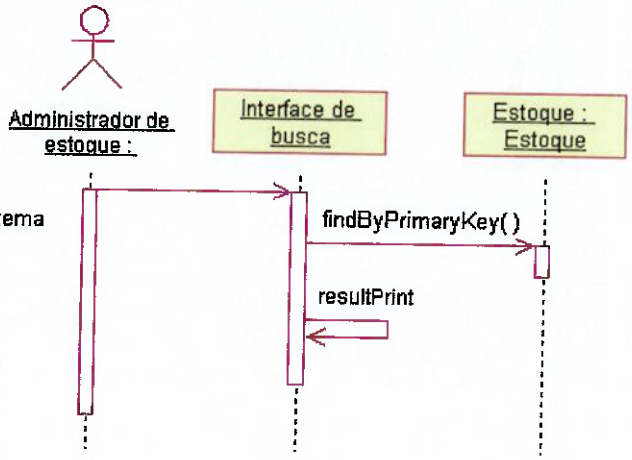


Remover estoque



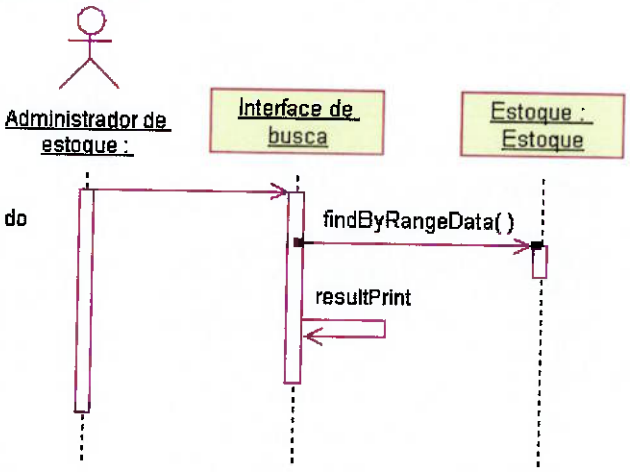
1. Para remover do estoque um item em uma data o operador fornece inicialmente ao sistema o código do item e a data.
2. O sistema retorna então o resultado da busca por esse registro com a opção de remover.
3. O sistema ainda pede uma confirmação ao usuário antes de realizar a operação.
4. Caso não se saiba o código do item há a possibilidade de realizar uma busca nos itens.

Consultar estoque por Item e Data



1. Para realizar a busca o operador fornece ao sistema o código do item e a data.
2. Caso não se saiba o código do item há a possibilidade de realizar uma busca nos itens.

Consultar estoque por Item e Data inicial e final

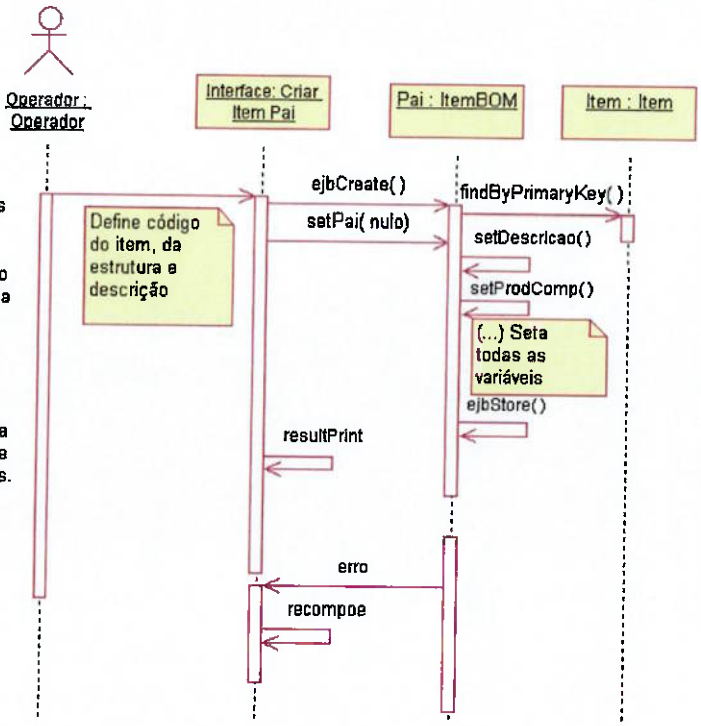


1. Para essa consulta o operador fornece o código do item e as datas inicial e final.
2. Caso não se saiba o código do item há a possibilidade de realizar uma busca nos itens.
3. O sistema irá montar uma tela com uma tabela horizontal de estoque dia a dia.

4.4.2 BOM – Estrutura de materiais

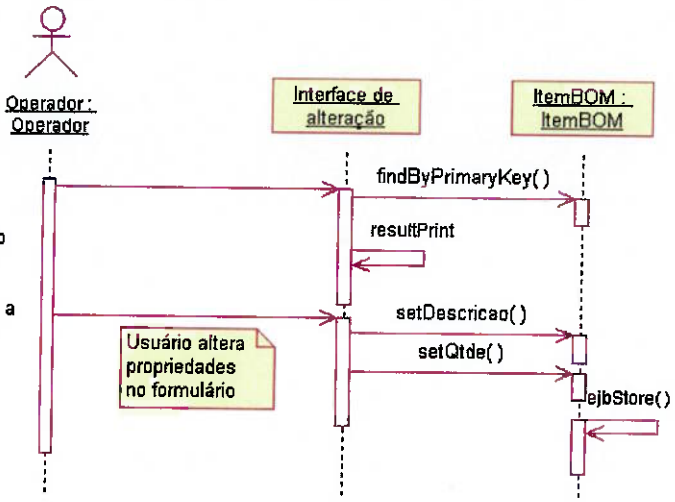
Criar item Pai

1. O operador define o código da estrutura (considerando-se que o mesmo item pai pode ter mais de uma estrutura), o código do item e uma descrição para a estrutura.
2. Na entrada dos dados o sistema verifica se o código do item é válido, se este é um "item final" (de demanda independente) e se ainda não existe o mesmo código para outra estrutura
3. Se estiver correto, o item é encontrado, todas as variáveis são setadas e é criado um item BOM com o atributo pai nulo.
4. Se não for um "item final", ou já existir o código para a estrutura o sistema retorna uma mensagem de erro e recompõe a tela para que o operador faça as correçoes.



Modificar item BOM

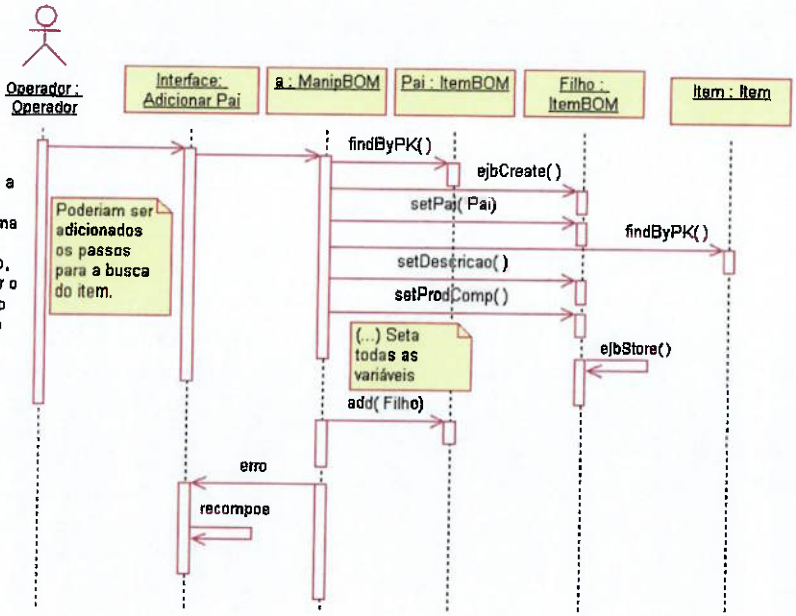
1. O operador fornece o código do item BOM.
2. O sistema monta um resultado com as características do item nessa estrutura e a informação se este item tem filhos, permitindo a alteração dos atributos descrição e qtde.
3. Se não for encontrado o item ou não for encontrada a estrutura o sistema informa o erro e recompõe a tela.
4. O atributo pai não pode ser alterado. Ele só é modificado quando o item é criado ou quando é adicionado ou removido de um Pai.



Adicionar item BOM

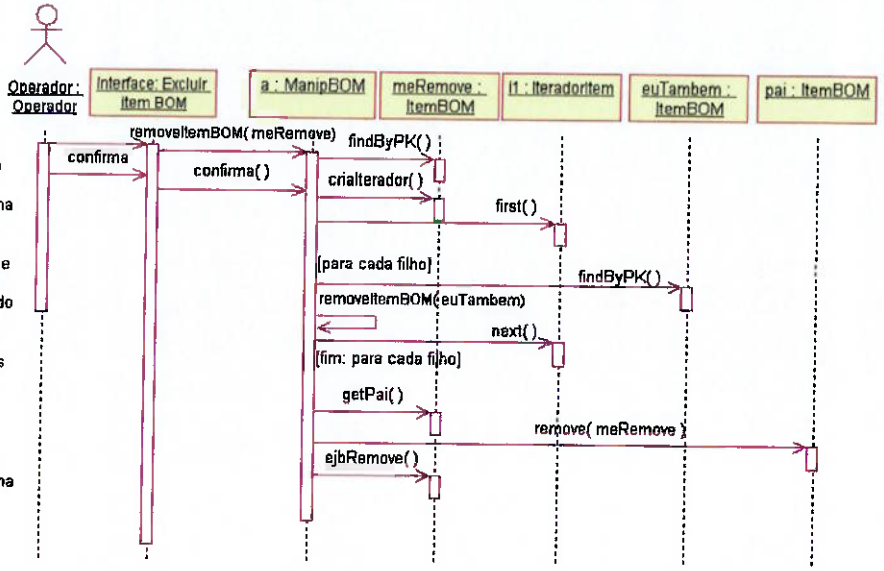
1. O operador entra o código do item BOM (ao qual será adicionado o item) e o código do item a ser adicionado.
3. O operador tem a possibilidade de realizar uma busca para escolha do item.
4. O sistema irá então validar o código fornecido, criar um item Filho com o atributo pai, encontrar o item correspondente e setar as variáveis no novo itemBOM criado, além de adicionar o novo Filho criado na tabela hash do Pai.

5. Se não for encontrado o item nessa estrutura ou não for encontrada a estrutura o sistema informa o erro e recompõe a tela.

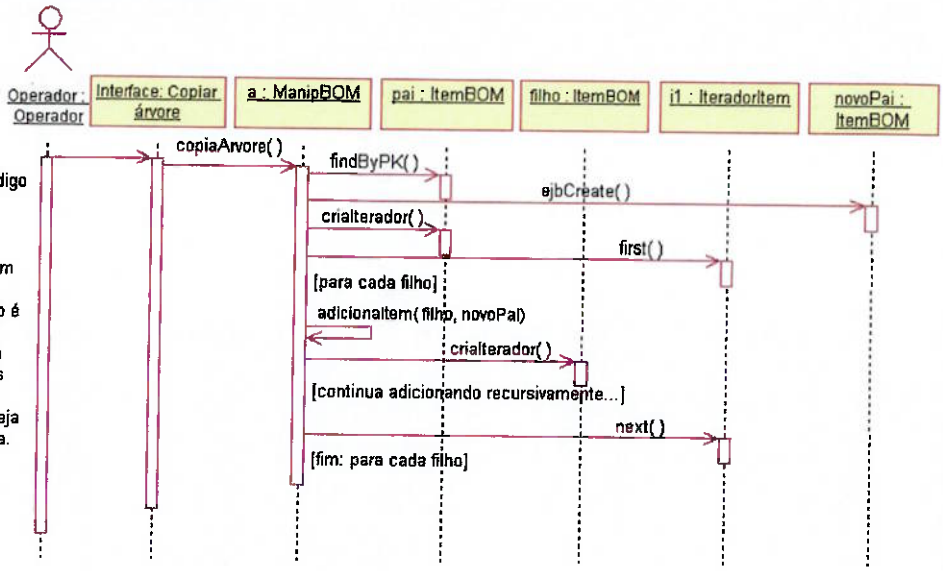


Excluir item BOM

1. O operador entra o código do item BOM a ser excluído.
2. Se o item for encontrado, o sistema verifica se tem filhos.
3. Se tem o sistema pede uma confirmação da exclusão deste item e de toda a árvore abaixo dele. Se não pede uma confirmação da exclusão do item.
4. Se a exclusão é confirmada, o sistema percorre toda a lista de filhos seguindo os passos:
4.1. Se o item não tem filhos ele é removido da tabela Hash do pai e apagado.
4.2. Se o item tem filhos, todos os filhos são apagados pela mesma rotina (recursivamente) e depois ele é removido da tabela hash do pai e apagado.

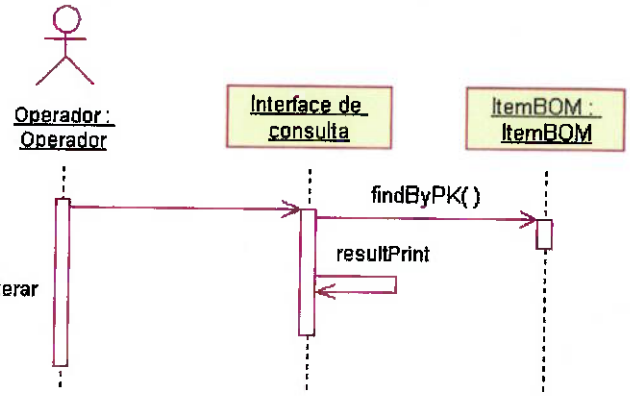


Copiar pai com todos os filhos



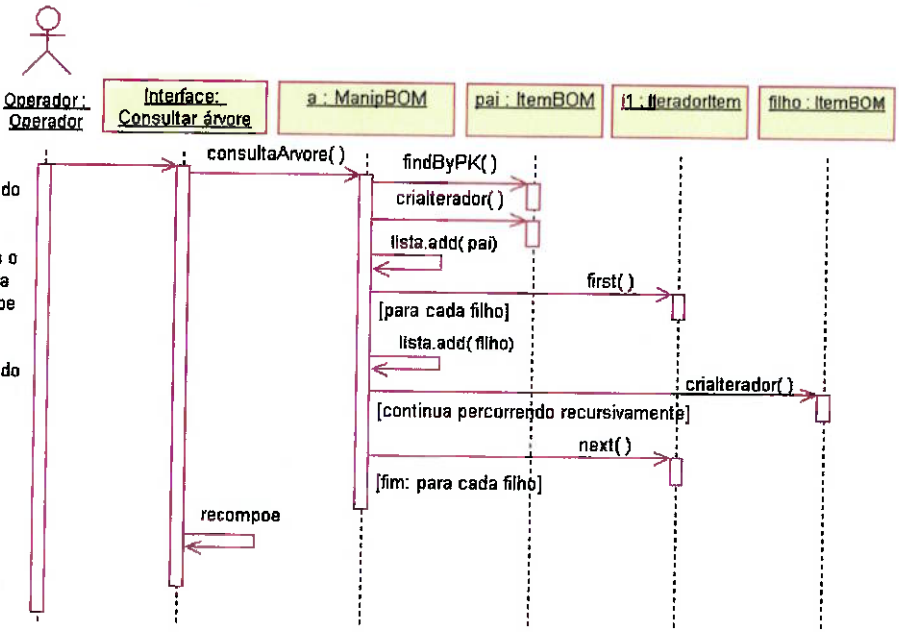
1. O operador entra o código do item BOM e o código para a nova estrutura.
2. Se o sistema não encontra o item retorna um erro e recompoe a tela.
3. Se o item é encontrado é criado um novo item pai com o novo código para a estrutura e todos os filhos são adicionados recursivamente até que seja percorrida toda a estrutura.

Consultar item na estrutura



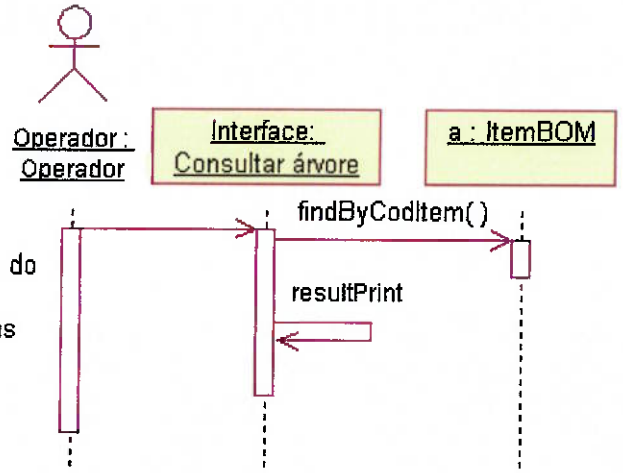
1. O operador fornece o código do item e o sistema retorna o item com suas características.
2. O operador tem a possibilidade de realizar uma consulta da árvore ou where-used, ou modificar ou alterar o item.

Consultar árvore abaixo de um pai



1. O operador entra o código do item BOM e o código da estrutura.
2. Se o sistema não encontra o item como pai nessa estrutura ele retorna um erro e recompoe a tela.
3. Se o item é encontrado a estrutura é percorrida a partir do item pai e a árvore é exibida.

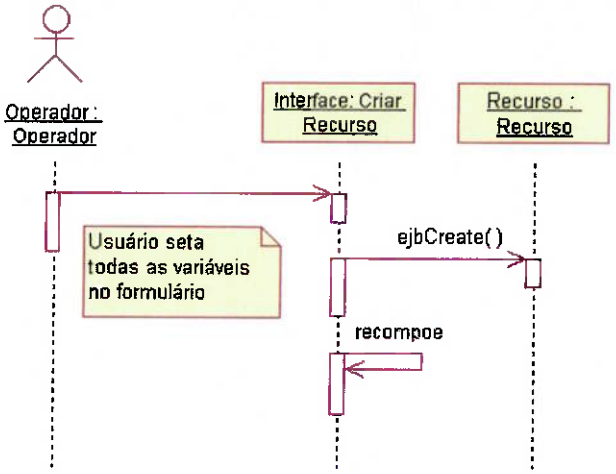
Consulta *where-used* de itens



1. O operador entra o código do item.
2. O sistema retorna todas as ocorrências desse item em todas as estruturas com os respectivos pais.

4.4.3 Processos

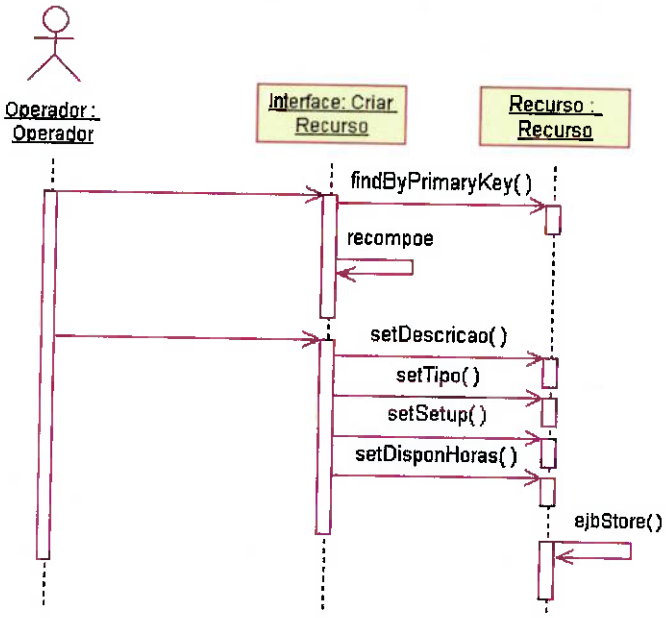
Criar recurso



1. O operador entra o código do recurso, a descrição, o tipo, o tempo de setup e disponibilidade de horas de uma unidade do recurso.
2. O sistema cria um novo recurso com a informação fornecida e mostra o resultado.
3. Caso o código já exista o sistema retorna um erro e recompoe a tela.
4. Todos os campos são obrigatórios. Para o setup se não é fornecido um valor o sistema assume zero.

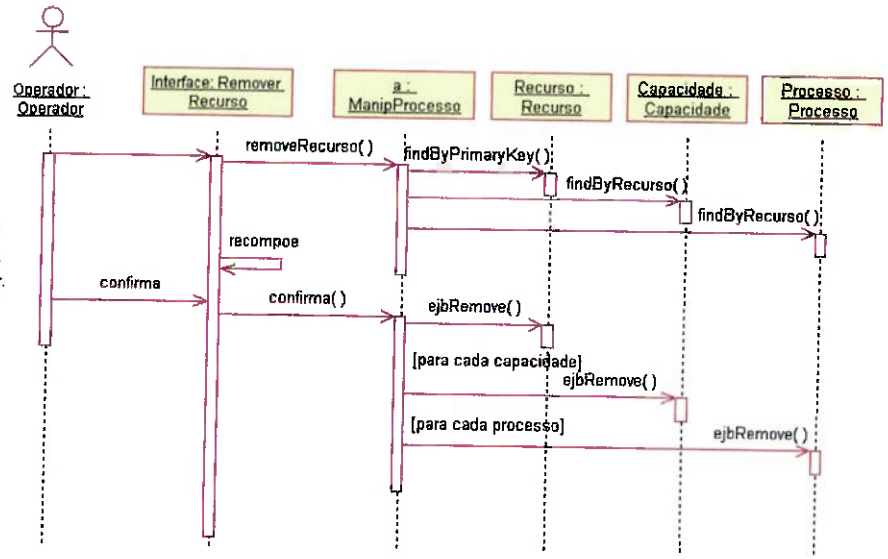
Alterar recurso

1. O operador entra o código do recurso a ser modificado.
2. Caso não seja encontrado o recurso, o sistema retorna um erro.
3. Do contrário o sistema monta uma interface para alteração da descrição, tipo, tempo de setup e disponibilidade de horas.
4. Todos os campos são obrigatórios. Para o setup se não é fornecido um valor o sistema assume zero.



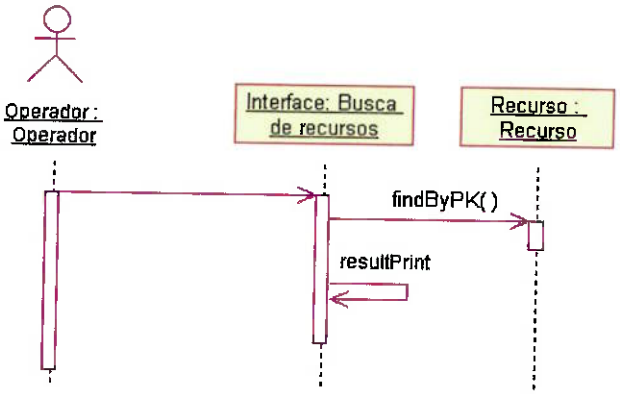
Remover recurso

1. O operador entra o código do recurso a ser removido.
2. Caso não seja encontrado o recurso, o sistema retorna um erro.
3. Do contrário o sistema retorna o recurso com suas características e pede confirmação antes de remover.
4. Se existirem processos e capacidade definida relativos ao recurso o sistema confirma a deleção de todas as instâncias.

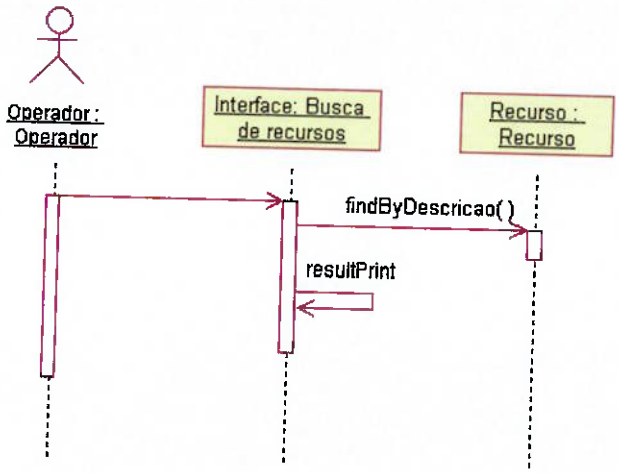


Consultar recursos por código

1. O operador entra o código do recurso e este é recuperado pelo sistema.
2. Com o recurso recuperado o operador tem a possibilidade de realizar as seguintes operações:
- Alteração/remoção do recurso
- Consulta de processos para o recurso
- Consulta de capacidade para o recurso

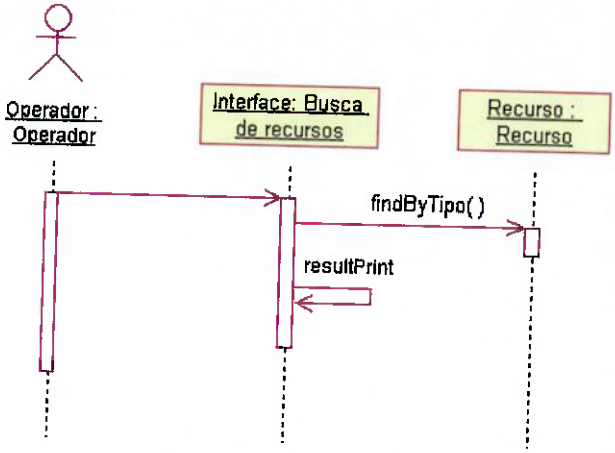


Consultar recursos por descrição



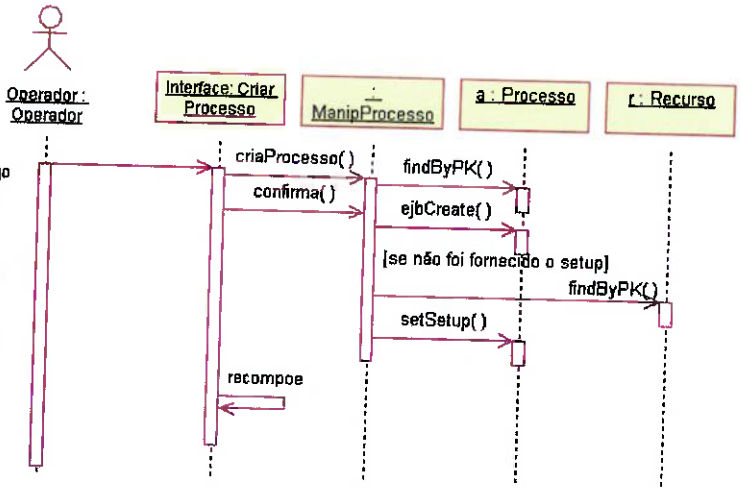
1. O operador entra a descrição ou parte dela. O sistema recupera todos os recursos cujas descrições contenham a string fornecida.
2. Os recursos serão exibidos em ordem alfabética de dez em dez.
3. Para cada recurso recuperado o operador tem a possibilidade de realizar as seguintes operações:
- Alteração/remoção do recurso
- Consulta de processos para o recurso
- Consulta de capacidade para o recurso

Consultar recursos por tipo



1. O operador entra o tipo do recurso e o sistema recupera todos os recursos desse tipo.
2. Os recursos serão exibidos em ordem alfabética de dez em dez.
3. Para cada recurso recuperado o operador tem a possibilidade de realizar as seguintes operações:
- Alteração/remoção do recurso
- Consulta de processos para o recurso
- Consulta de capacidade para o recurso

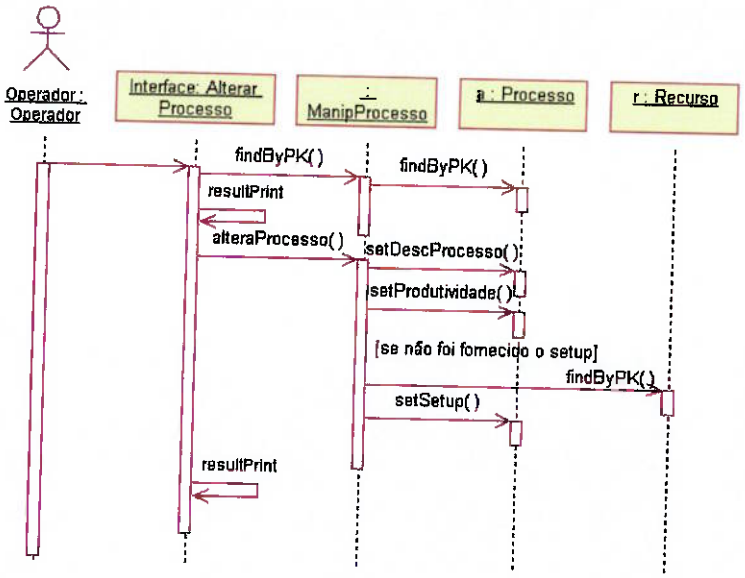
Criar processo



1. O operador entra o código do item BOM, o código do recurso, a descrição, a produtividade e o tempo de setup do processo.
2. O sistema cria um novo processo com a informação fornecida e mostra o resultado.
3. Caso já exista um processo para este item BOM e este recurso o sistema pergunta se o operador quer substituir o processo.
4. Todos os campos são obrigatórios. Para o setup se não é fornecido um valor o sistema assume o setup do recurso.
5. O operador tem a possibilidade de realizar uma busca na estrutura e nos recursos para definir o código.

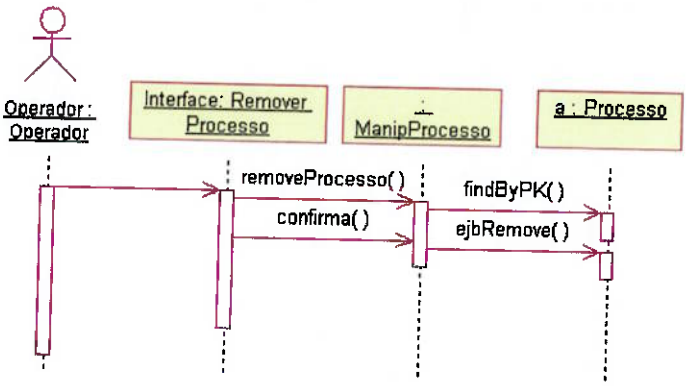
Alterar processo

1. O operador entra o código do item BOM e o código do recurso para o processo a ser alterado.
2. Caso não exista um processo para este item BOM e este recurso o sistema retorna um erro.
3. Do contrário o sistema monta um resultado com o processo especificado permitindo a edição da descrição, setup e produtividade.
4. Todos os campos são obrigatórios. Para o setup se não é fornecido um valor o sistema assume o setup do recurso.
5. O operador tem a possibilidade de realizar uma busca na estrutura e nos recursos para definir o código.



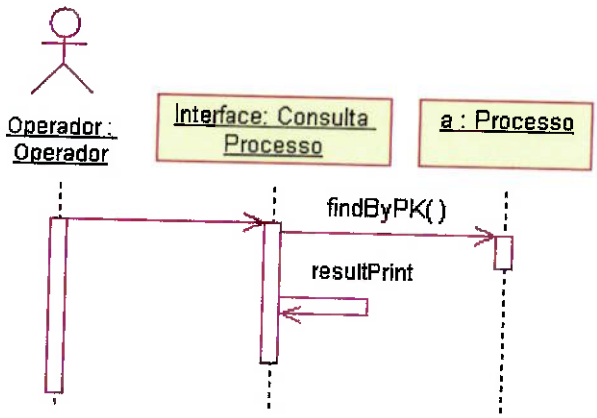
Remover processo

1. O operador entra o código do item BOM e o código do recurso para o processo a ser removido.
2. Caso não exista um processo para este item BOM e este recurso o sistema retorna um erro.
3. Do contrário o sistema pede uma confirmação antes de remover o processo.

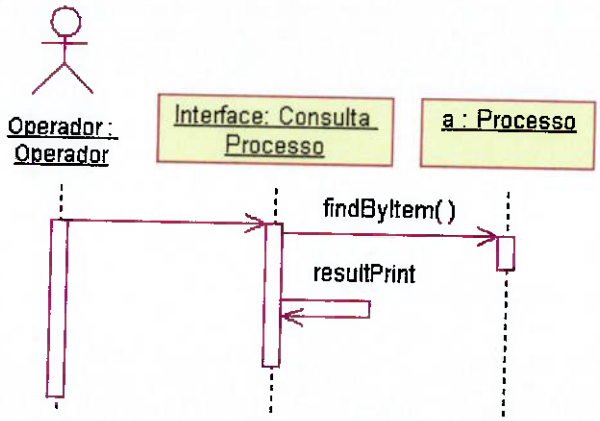


Consultar processos por Item/Recurso

1. O operador entra o código do item BOM e o código do recurso para o processo..
2. Caso não exista um processo para este item BOM e este recurso o sistema retorna um erro.
3. Do contrário o sistema monta um resultado com o processo.

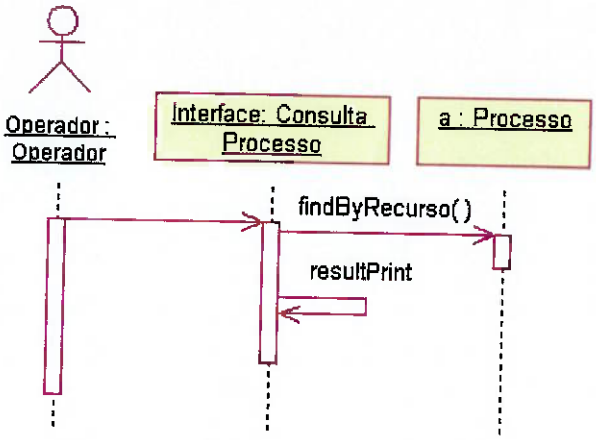


Consultar processos por Item



1. O operador entra o código do item BOM.
2. Caso não exista um processo para este item BOM sistema retorna um erro.
3. Do contrário o sistema monta um resultado com os processos encontrados.

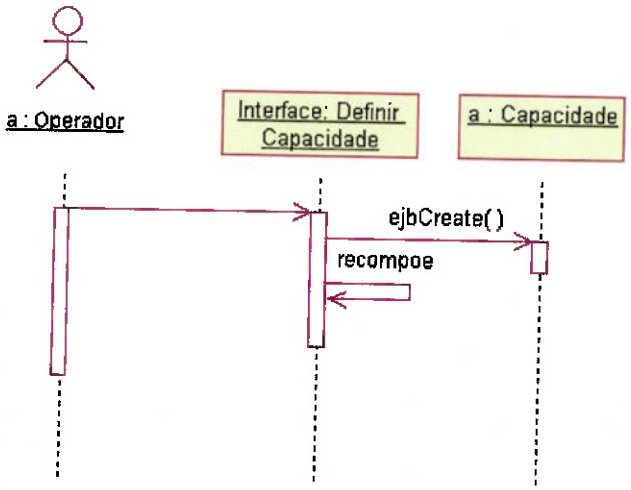
Consultar processos por Recurso



1. O operador entra o código do recurso.
2. Caso não exista um processo para este recurso sistema retorna um erro.
3. Do contrário o sistema monta um resultado com os processos encontrados.
4. Os processos serão exibidos de dez em dez ordenados por item BOM.

4.4.4 Capacidade

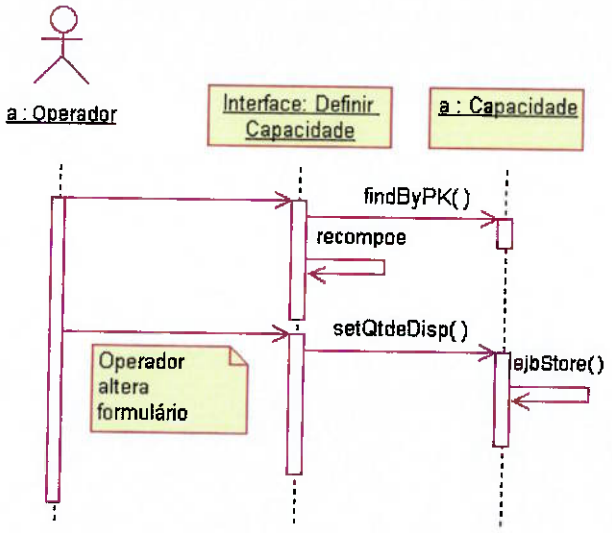
Definir capacidade



1. O operador fornece ao sistema o código do recurso, a data, e a quantidade disponível do recurso.
2. Caso não saiba o código do recurso ele tem a possibilidade de realizar uma busca.
3. Se o código ou a data forem inválidos o sistema retorna uma mensagem de inconsistência e volta para a tela anterior com os dados enviados.
4. Todos os outros dados referentes à capacidade nessa data são calculados pelo sistema e aparecem no formulário como campos desabilitados.

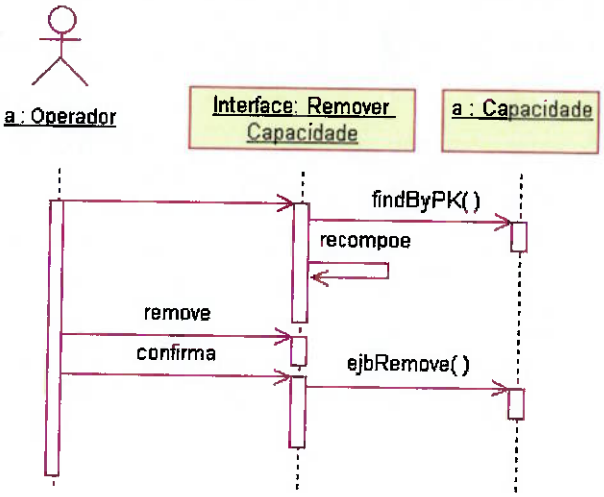
Alterar capacidade

1. O operador fornece ao sistema o código do recurso e a data.
2. Caso não saiba o código do recurso ele tem a possibilidade de realizar uma busca.
3. Se o código ou a data forem inválidos o sistema retorna uma mensagem de inconsistência.
4. Do contrário monta um formulário com o resultado da busca permitindo a edição do campo quantidade disponível.



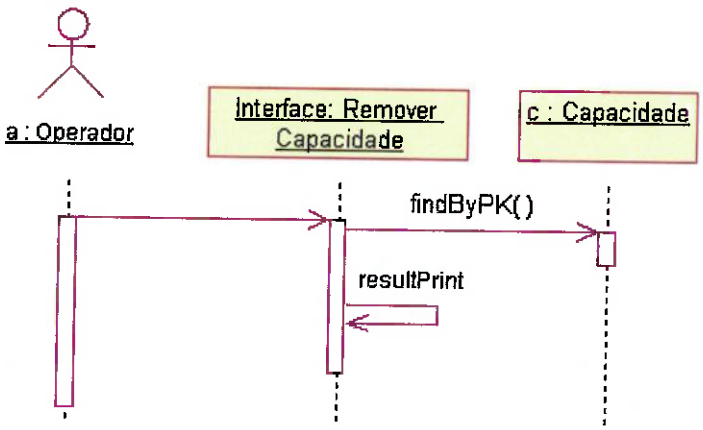
Remover capacidade

1. O operador fornece ao sistema o código do recurso e a data.
2. Caso não saiba o código do recurso ele tem a possibilidade de realizar uma busca.
3. O sistema ainda pede uma confirmação antes de excluir a capacidade

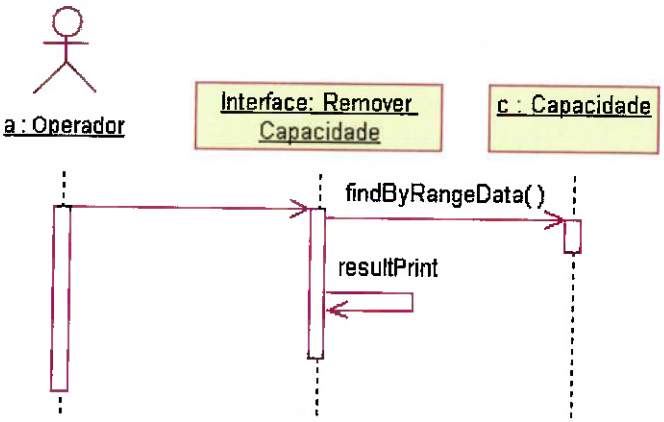


Consultar capacidade por recurso e data

1. O operador entra o código do recurso e a data.
2. Caso não saiba o código do recurso ele tem a possibilidade de realizar uma busca.



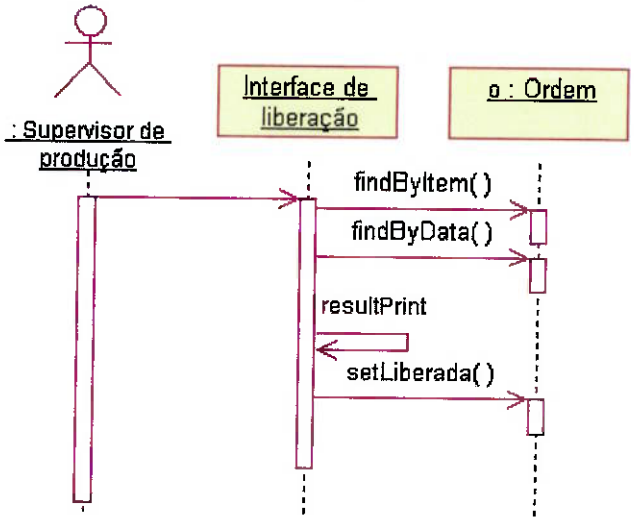
Consultar capacidade por recurso e data inicial e final



1. O operador entra o código do recurso e as datas inicial e final.
2. Caso não saiba o código do recurso ele tem a possibilidade de realizar uma busca.
3. O sistema monta o resultado na forma de uma tabela horizontal mostrando a capacidade dia a dia.

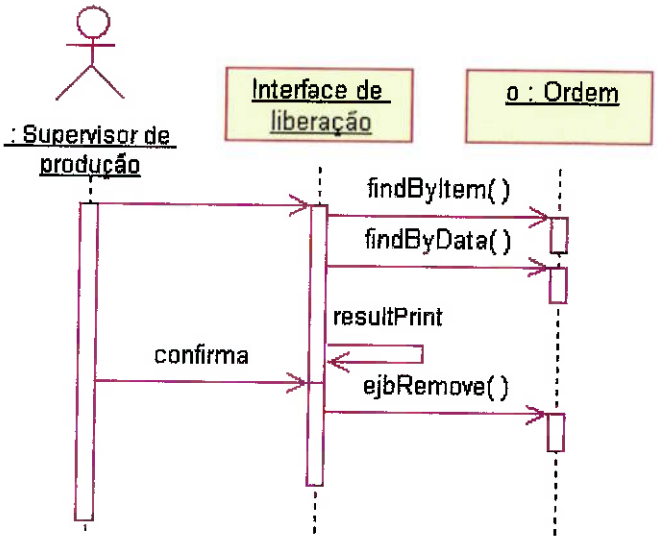
4.4.5 Ordens

Liberar ordem



1. O supervisor realiza uma busca que traz as ordens referentes à busca com a possibilidade de liberar cada uma.
2. Se a ordem for liberada o sistema promove a alteração do recebimento programado na tabela de estoque.

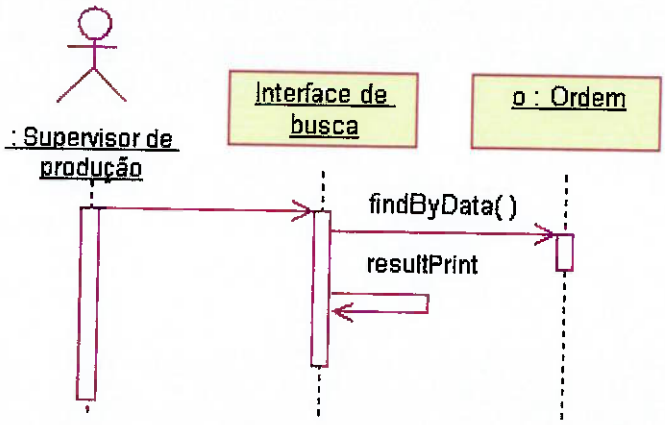
Remover ordem



1. O supervisor realiza uma busca que traz as ordens referentes à busca com a possibilidade de remover cada uma.
2. O sistema ainda pede uma confirmação antes de remover a ordem.

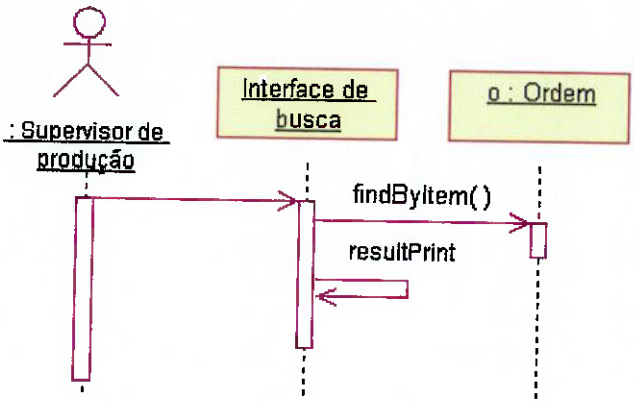
Consultar ordens por data

1. O supervisor entra a data e o sistema monta o resultado com todas as ordens colocadas nessa data.



Consultar ordens por item

1. O supervisor entra o código do item e o sistema monta o resultado com todas as ordens colocadas para esse item.
2. Ele ainda tem a possibilidade de realizar uma busca para encontrar o item.



4.5 Requisitos Não-Funcionais

4.5.1 Ambiente de execução

O ambiente de teste está utilizando um PC com PentiumIII com 256 Mb de RAM. Esse ambiente é, na verdade, apropriado para o desenvolvimento, já que se estão utilizando ferramentas relativamente pesadas como o ambiente de desenvolvimento integrado (IDE – *integrated Development* Environment) **Forte®** da Sun e a ferramenta CASE (*Computer Aided Sotware Engineering*) **Rational Rose®**.

Alguns testes com componentes também foram realizados nesse ambiente e o resultado a princípio é satisfatório. É claro que um negócio com volume muito grande de itens e grandes árvores de materiais irá demandar um ambiente robusto.

O que se pode prever, a princípio, é que o gargalo do sistema será o cálculo completo do MRP, mais especificamente no que diz respeito às operações no banco de dados. Assim se faz necessária uma boa ferramenta de banco de dados e, dependendo do volume do negócio, um equipamento robusto para comportar esse sistema. Entre as ferramentas consideradas previamente estão os SGBDR (Sistema Gerenciadores de Banco de Dados

Relacionais) mais utilizados no mercado: Oracle, IBM DB2, SQL Server da Microsoft, Informix da IBM, Sybase e NCR, entre outros. No ambiente de teste estará sendo utilizado o **SQL Server** da Microsoft.

A próxima ferramenta considerada e de fundamental importância no projeto é o servidor de aplicações (ver item 3.2.2). Nesse projeto todos os testes e implementação farão uso do **JBoss**, que é uma ferramenta de código aberto. Não faz parte do escopo desse trabalho um estudo desse mercado ou a comparação de ferramentas. No entanto cabe dizer que em uma análise preliminar de custo benefício seria difícil bater uma ferramenta gratuita.

O sistema operacional utilizado nos testes é o **Windows 2000**. Em uma análise preliminar consideramos que um servidor baseado em PC/Intel, desde que conte com boa infra-estrutura (com Arrays de disco SCSI, por exemplo, além de memória rápida e em grande volume, bom cache de memória etc.) pode ser apropriado para aplicações com volume médio[1] (da ordem de centenas de itens). Aplicações com maior volume provavelmente exigiriam equipamentos mais robustos (como máquinas RISC) provavelmente em sistema operacional UNIX.

Além das ferramentas de desenvolvimento utilizadas e do sistema operacional, SGBDR e servidor de aplicação, foram utilizadas nos testes ferramentas de código aberto marginais. São elas:

- **Ant**: ferramenta para compilar e implementar software depois de escrito, voltado para J2SE e J2EE.
- **XDocLet**: ferramenta para gerar arquivos complementares do JBoss auxiliando na confecção dos EJB.
- **JavaService**: ferramenta para executar o JBoss (e outras sistemas escritos em java) como serviços do Windows.

4.5.2 Interfaces com outros sistemas

A plataforma J2EE tem extensa documentação sobre a forma como sistemas nesse padrão podem acessar outros sistemas legados. Esse projeto não tem em seu escopo a realização desse tipo de teste. No entanto a própria utilização de um banco de dados relacional aberto é uma forma de realizar essa interface.

A principal interface, portanto, é a realizada com o banco de dados (Microsoft SQL Server®, no caso), o que é realizado por meio de JDBC (*Java Database Connectivity*).

---

[1] Essa é uma suposição não testada.

4.5.3 Desempenho e Escalabilidade

O sistema mostra desempenho satisfatório nos testes mas em um ambiente e negócio mais robustos deve apresentar requisitos de hardware também mais robustos. Como o projeto é feito com utilização de componentes e código aberto e utiliza linguagem java que é multi-plataforma, a escalabilidade não deve ser um problema.

4.5.4 Segurança e Privacidade

A segurança do sistema não entrou no escopo desse projeto.

4.6 Perspectiva Gerencial

4.6.1 Primeira parte

	Jul	Ago	Set	Out	Nov
Familiarização com linguagem Java					
Pesquisa EJB					
Pesquisa UML					
Especificação do projeto					
Modelagem do sistema					

4.6.2 Segunda parte

	Fev	Mar	Abr	Mai	Jun
Revisão da especificação					
Revisão da modelagem do sistema					
Implementação					
Testes e ajustes					
Redação do relatório final					

5 Implementação e testes

Nesse capítulo iremos apresentar toda a implementação e testes do sistema a partir do modelo desenvolvido e apresentado no capítulo anterior. Vamos iniciar com uma descrição da aplicação de forma geral e seguir com a implementação de cada componente dividida pelos *packages* discutidos no item 4.2.

Optamos por mostrar o código apenas para um Entity Bean como exemplo já que o código completo estará disponível na interface da própria aplicação e em um CD anexo.

5.1 Aplicação

Como já discutido, toda a interface com o sistema será feita por meio de páginas JSP. Dessa forma a interface para o cliente será na verdade um navegador (*browser*) de Internet. Estamos utilisando o Internet Explorer® para os testes.

A janela inicial do sistema é mostrada na Figura 25.

No cabeçalho existem três botões o primeiro (Início) volta ao início do site (ou essa mesma janela), o segundo (ERP) é da aplicação em si e o terceiro mostra o console do JBoss e serve para administrar e consultar o *container*.

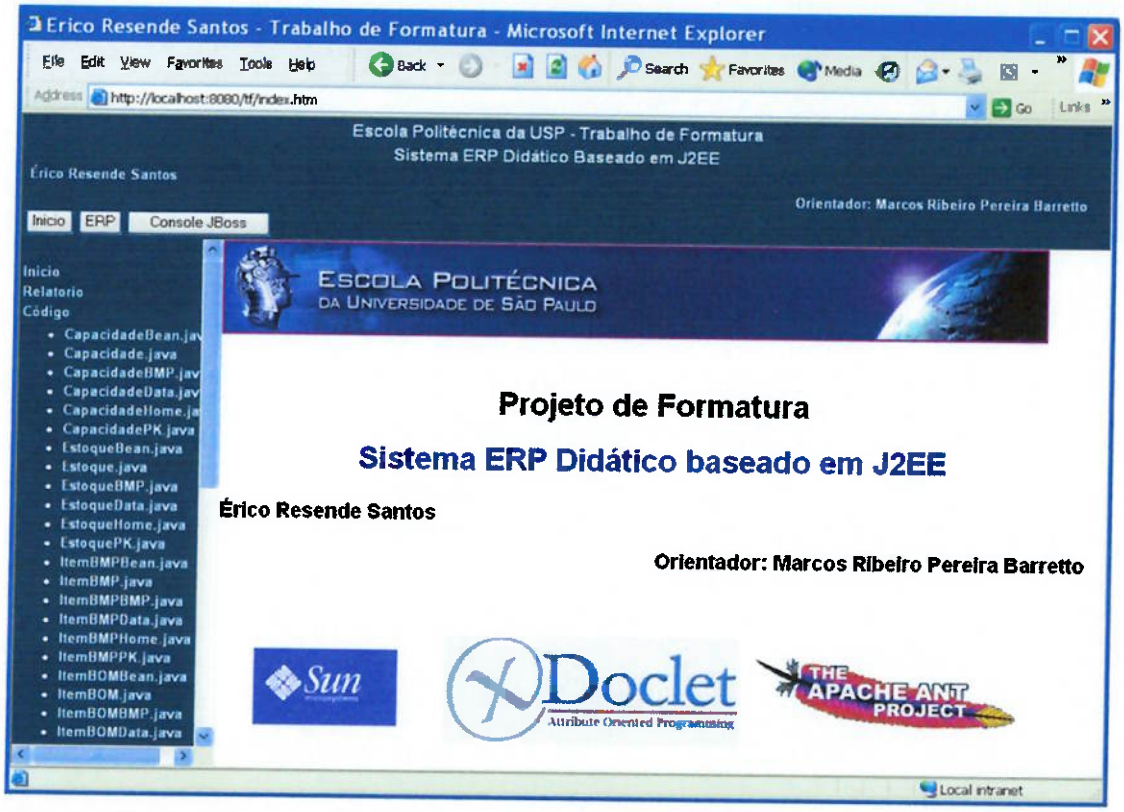


Figura 25. Tela inicial do sistema

Na janela do início existe um menu à esquerda e uma janela à direita para o conteúdo. Nessa porção do sistema estarão disponíveis este relatório, os códigos fonte, outros links diversos e um link para contato por e-mail. Trata-se de uma aplicação paralela para descrever a aplicação principal e executar a parte didática do objetivo desse projeto.

A janela inicial da aplicação (acessível pelo segundo botão do cabeçalho) é mostrada na Figura 26.

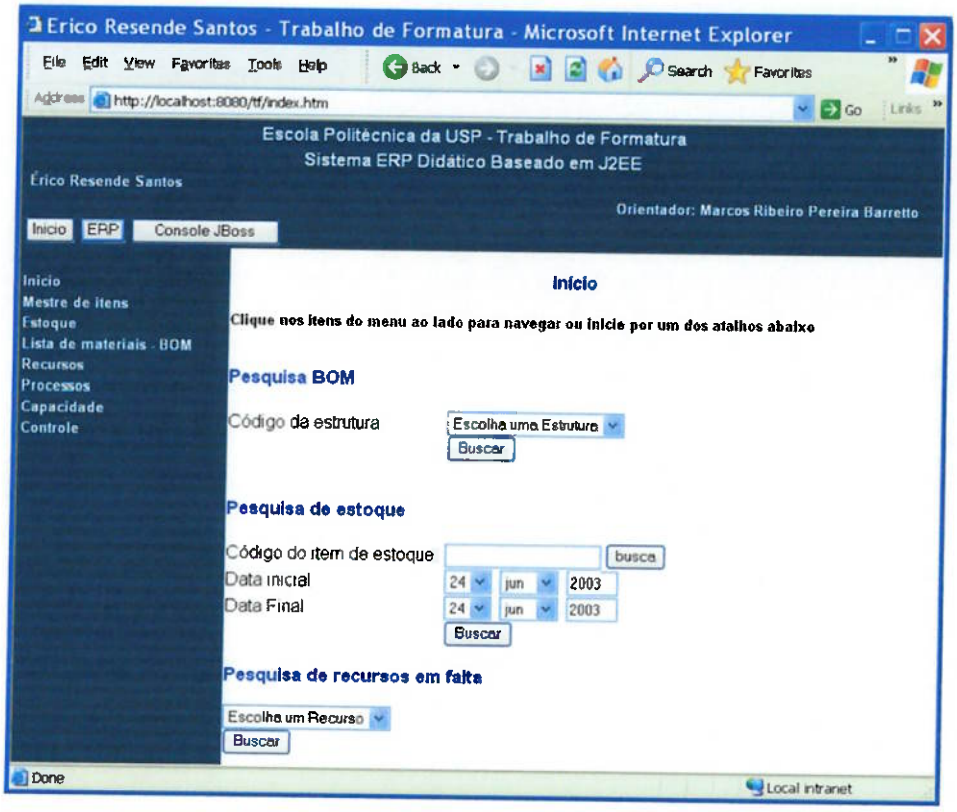


Figura 26. Página inical da aplicação

O usuário pode selecionar alguma função no menu à esquerda ou começar por um dos atalhos para consultas mais frequentes. No menu estão disponíveis acessos para todos os módulos da aplicação. Passaremos então a apresentar cada um desses módulos.

5.2 Estoque

Foram implementados dois Entity Bens para essa *package*. Um para itens e um para o estoque (ver Figura 24).

5.2.1 Itens

A criação/alteração/remoção/consulta de itens foi realizada por meio de um Entity Bean (servidor) e uma página JSP (cliente), além de uma tabela no banco de dados para a persistência.

Para este Entity Bean/cliente JSP estão apresentados a seguir os arquivos que compõe o código. Como já discutido, incluímos apenas esse código no relatório para fins didáticos. O resto do código se encontra junto à aplicação e em um CD anexo.

A seguir é mostrado o funcionamento do EJB para o nosso exemplo. Primeiramente trazemos a classe ItemBMPBean. Trata-se da principal classe e através da qual são geradas as outras. Percebe-se que ela implementa a interface EntityBean.

```
/*
* Entity Bean (BMP) para itens de estoque
*
*/
package MRP;

import java.sql.*;
import java.util.*;
import javax.ejb.*;
import javax.naming.*;
import MRP.ItemBMPPK;

/**
 * Entity Bean (BMP) para itens de estoque
 *
 * @author Erico Resende Santos
 *
 * @ejb:bean name="MRP/ItemBMP"
 *           display-name="Entity Bean (BMP) para itens de estoque"
 *           type="BMP"
 *           jndi-name="ejb/MRP/ItemBMP"
 *
 **/
public abstract class ItemBMPBean implements EntityBean
{

   public EntityContext ctx;

    // ------------------------------------------------------------------------
    // Campos
    // ------------------------------------------------------------------------
    /*
    private String codigo; // PK
    private String descricao;
    private String prodComp;
    private String finalInter;
    private String realFant;
    private boolean calcMRP;
    private String unidade;
    private double custo;
    private double qtdeLote;
    private double leadTime;
    private int LLC;
    private double estSeguranca;
    private java.sql.Date dataAlt;
    private String altPor;
    */
    String codigo; // PK
    String descricao;
    String prodComp;
    String finalInter;
    String realFant;
    boolean calcMRP;
    String unidade;
```

```
    double custo;
    double qtdeLote;
    double leadTime;
    int LLC;
    double estSeguranca;
    java.sql.Date dataAlt;
    String altPor;

    public ItemBMPBean() {
        System.out.println("Novo EJBObject <ItemBMPBean> sendo criado");
    }

    // Getter/Setter

   /**
   * Retorna o código
   * @return Código do item
   * @ejb:persistent-field
   * @ejb:pk-field
   * @ejb:interface-method view-type="remote"
   **/
   public String getCodigo() { return codigo; }

   /**
   * Seta o código
   * @param pCodigo The id of this TestEntity. Is set at creation time.
   * @ejb:interface-method view-type="remote"
   **/
   public void setCodigo( String pCodigo ) { this.codigo = pCodigo; }

   /**
   * Retorna a descrição
   * @return Descrição
   * @ejb:persistent-field
   * @ejb:interface-method view-type="remote"
   **/
   public String getDescricao() { return descricao; }

   /**
   * Seta a Descrição
   * @param pDescricao
   * @ejb:interface-method view-type="remote"
   **/
   public void setDescricao(String pDescricao) { this.descricao = pDescricao; }

   /**
   * Retorna prodComp (produzido-"P" ou comprado "C")
   * @return prodComp
   * @ejb:persistent-field
   * @ejb:interface-method view-type="remote"
   **/
   public String getProdComp() { return prodComp; }

   /**
   * Seta prodComp
   * @param pProdComp
   * @ejb:interface-method view-type="remote"
   **/
   public void setProdComp(String pProdComp) { this.prodComp = pProdComp; }

   /**
   * Retorna finalInter (final-"F" ou Intermediário "I")
   * @return finalInter
   * @ejb:persistent-field
   * @ejb:interface-method view-type="remote"
   **/
```

```
    public String getFinalInter() { return finalInter; }

    /**
    * Seta finalInter
    * @param pfinalInter
    * @ejb:interface-method view-type="remote"
    **/
    public void setFinalInter(String pfinalInter) { this.finalInter = pfinalInter; }

    /**
    * Retorna realFant (Real-"R" ou Fantasia "F")
    * @return realFant
    * @ejb:persistent-field
    * @ejb:interface-method view-type="remote"
    **/
    public String getRealFant() { return realFant; }

    /**
    * Seta realFant
    * @param pRealFant
    * @ejb:interface-method view-type="remote"
    **/
    public void setRealFant(String pRealFant) { this.realFant = pRealFant; }

    /**
    * Retorna calcMRP (true se for calculado por MRP)
    * @return calcMRP
    * @ejb:persistent-field
    * @ejb:interface-method view-type="remote"
    **/
    public boolean getCalcMRP() { return calcMRP; }

    /**
    * Seta realFant
    * @param pRealFant
    * @ejb:interface-method view-type="remote"
    **/
    public void setCalcMRP(boolean pCalcMRP) {
        this.calcMRP = pCalcMRP;
    }

    /**
    * Retorna a unidade (m, KG, ml etc.)
    * @return unidade
    * @ejb:persistent-field
    * @ejb:interface-method view-type="remote"
    **/
    public String getUnidade() { return unidade; }

    /**
    * Seta a unidade
    * @param pUnidade
    * @ejb:interface-method view-type="remote"
    **/
    public void setUnidade(String pUnidade) { this.unidade = pUnidade; }

    /**
    * Retorna o custo
    * @return custo
    * @ejb:persistent-field
    * @ejb:interface-method view-type="remote"
    **/
    public double getCusto() { return custo; }

    /**
    * Seta o custo
```

```
     * @param pCusto
     * @ejb:interface-method view-type="remote"
     **/
     public void setCusto(double pCusto) { this.custo = pCusto; }

     /**
     * Retorna o lote
     * @return qtdeLote
     * @ejb:persistent-field
     * @ejb:interface-method view-type="remote"
     **/
     public double getQtdeLote() { return qtdeLote; }

     /**
     * Seta qtdeLote
     * @param pQtdeLote
     * @ejb:interface-method view-type="remote"
     **/
     public void setQtdeLote(double pQtdeLote) { this.qtdeLote = pQtdeLote; }

     /**
     * Retorna o Lead Time
     * @return leadTime
     * @ejb:persistent-field
     * @ejb:interface-method view-type="remote"
     **/
     public double getLeadTime() { return leadTime; }

     /**
     * Seta leadTime
     * @param pLeadTime
     * @ejb:interface-method view-type="remote"
     **/
     public void setLeadTime(double pLeadTime) { this.leadTime = pLeadTime; }

     /**
     * Retorna o LLC (Low Level Code)
     * @return LLC
     * @ejb:persistent-field
     * @ejb:interface-method view-type="remote"
     **/
     public int getLLC() { return LLC; }

     /**
     * Retorna o Estoque de Segurança
     * @return estSeguranca
     * @ejb:persistent-field
     * @ejb:interface-method view-type="remote"
     **/
     public double getEstSeguranca() { return estSeguranca; }

     /**
     * Seta estSeguranca
     * @param pEstSeguranca
     * @ejb:interface-method view-type="remote"
     **/
     public void setEstSeguranca(double pEstSeguranca) { this.estSeguranca =
pEstSeguranca; }

     /**
     * Retorna a data de alteração
     * @return dataAlt
     * @ejb:persistent-field
     * @ejb:interface-method view-type="remote"
     **/
     public java.sql.Date getDataAlt() { return dataAlt; }
```

```
    /**
     * Retorna Alterado por?
     * @return altPor
     * @ejb:persistent-field
     * @ejb:interface-method view-type="remote"
     **/
    public String getAltPor() { return altPor; }

    /**
     * Atualiza o EJB
     * @ejb:interface-method view-type="remote"
     **/
    public void atualiza() {
        System.out.println("Chamando atualiza() em ItemBMPBean: "+this.codigo);
        ItemBMPPK pk = (ItemBMPPK) ctx.getPrimaryKey();
        this.codigo = pk.codigo;
        Connection con = null;
        PreparedStatement query = null;
        try {
            con = getConnection();
            query = con.prepareStatement("select * from itens where codigo = ?");
            query.setString(1, this.codigo);
            ResultSet rs = query.executeQuery();
            rs.next();
            this.descricao = rs.getString("descricao").trim();
            this.prodComp = rs.getString("prodComp").trim();
            this.finalInter = rs.getString("finalInter").trim();
            this.realFant = rs.getString("realFant").trim();
            this.calcMRP = rs.getBoolean("calcMRP");
            this.unidade = rs.getString("unidade").trim();
            this.custo = rs.getDouble("custo");
            this.qtdeLote = rs.getDouble("qtdeLote");
            this.leadTime = rs.getDouble("leadTime");
            this.estSeguranca = rs.getDouble("estSeguranca");
            this.dataAlt = rs.getDate("dataAlt");
            this.altPor = rs.getString("altPor").trim();
        }
        catch (Exception e) {
            throw new EJBException("Não foi possível recuperar o item "+pk, e);
        }
        finally {
            try { if (query != null) query.close(); }
            catch (Exception e) {}
            try { if (con != null) con.close(); }
            catch (Exception e) {}
        }
    }

    /**
     * Faz conexão com o banco
     * @return JDBC connection
     */
    public Connection getConnection() throws Exception {
        try {
            Connection con = null;
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            con = DriverManager.getConnection("jdbc:odbc:TF", "sa", "naish90");
            con.setAutoCommit(true);
            return con;
        }
        catch (Exception e) {
            e.printStackTrace();
            throw e;
        }
    }
```

```
    // ---------------------------------------------------------------------
    // EJB-required methods
    // ---------------------------------------------------------------------

   /**
    * Cria um item e insere no banco de dados
    * @param codigo, descricao, prodComp, finalInter, realFant,
    *        calcMRP, unidade, custo, qtdeLote, leadTime, estSeguranca
    * @throws CreateException
    * @return chave primária
    * @ejb:create-method view-type="remote"
    **/
    public ItemBMPPK ejbCreate( String codigo, String descricao, String prodComp,
                                String finalInter, String realFant, boolean
calcMRP,
                                String unidade, double custo, double qtdeLote,
                                double leadTime, double estSeguranca)
    throws CreateException {
        System.out.println("Tentando criar "+descricao);
        Connection con = null;
        PreparedStatement query = null;
        this.codigo = codigo;
        this.descricao = descricao;
        this.prodComp = prodComp;
        this.finalInter = finalInter;
        this.realFant = realFant;
        this.calcMRP = calcMRP;
        this.unidade = unidade;
        this.custo = custo;
        this.qtdeLote = qtdeLote;
        this.leadTime = leadTime;
        this.estSeguranca = estSeguranca;
        this.dataAlt = new java.sql.Date(System.currentTimeMillis());
        System.out.println(this.dataAlt);
        this.altPor = "Admin";
        System.out.println(this.altPor);
        try {
            con = getConnection();
            query = con.prepareStatement("insert into itens (codigo, descricao,
prodComp, "
                + "finalInter, realFant, calcMRP, unidade, custo, qtdeLote,
leadTime, "
                + "estSeguranca, dataAlt, altPor) values
(?,?,?,?,?,?,?,?,?,?,?,?,?)");
            query.setString(1, codigo);
            query.setString(2, descricao);
            query.setString(3, prodComp);
            query.setString(4, finalInter);
            query.setString(5, realFant);
            query.setBoolean(6, calcMRP);
            query.setString(7, unidade);
            query.setDouble(8, custo);
            query.setDouble(9, qtdeLote);
            query.setDouble(10, leadTime);
            query.setDouble(11, estSeguranca);
            query.setString(12, new java.text.SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(this.dataAlt));
            query.setString(13, this.altPor);
            query.executeUpdate();
            return new ItemBMPPK(this.codigo);
        }
        catch(Exception e) {
            throw new CreateException(e.toString());
        }
        finally {
```

```
            try { if (query != null) query.close(); }
            catch (Exception e) {}
            try { if (con != null) con.close(); }
            catch (Exception e) {}
        }
    }
    public void ejbPostCreate( String codigo, String descricao, String prodComp,
                               String finalInter, String realFant, boolean
calcMRP,
                               String unidade, double custo, double qtdeLote,
                               double leadTime, double estSeguranca) {}

    public void setEntityContext( EntityContext lContext ) {
        this.ctx = lContext;
    }

    public void unsetEntityContext() {
        ctx = null;
    }

    public void ejbActivate() {
        System.out.println("Chamando ejbActivate() em ItemBMPBean");
    }

    public void ejbPassivate() {
        System.out.println("Chamando ejbPassivate() em ItemBMPBean");
    }

   public void ejbLoad() {
        System.out.println("Chamando ejbLoad() em ItemBMPBean: "+this.codigo);
        ItemBMPPK pk = (ItemBMPPK) ctx.getPrimaryKey();
        this.codigo = pk.codigo;
        Connection con = null;
        PreparedStatement query = null;
        try {
            con = getConnection();
            query = con.prepareStatement("select * from itens where codigo = ?");
            query.setString(1, this.codigo);
            ResultSet rs = query.executeQuery();
            rs.next();
            this.descricao = rs.getString("descricao").trim();
            this.prodComp = rs.getString("prodComp").trim();
            this.finalInter = rs.getString("finalInter").trim();
            this.realFant = rs.getString("realFant").trim();
            this.calcMRP = rs.getBoolean("calcMRP");
            this.unidade = rs.getString("unidade").trim();
            this.custo = rs.getDouble("custo");
            this.qtdeLote = rs.getDouble("qtdeLote");
            this.leadTime = rs.getDouble("leadTime");
            this.estSeguranca = rs.getDouble("estSeguranca");
            this.dataAlt = rs.getDate("dataAlt");
            this.altPor = rs.getString("altPor").trim();
        }
        catch (Exception e) {
            throw new EJBException("Não foi possível recuperar o item "+pk, e);
        }
        finally {
            try { if (query != null) query.close(); }
            catch (Exception e) {}
            try { if (con != null) con.close(); }
            catch (Exception e) {}
        }
   }

   public void ejbStore() {
        //System.out.println("Chamando ejbStore() em ItemBMPBean");
```

```
        Connection con = null;
        PreparedStatement query = null;
        this.dataAlt = new java.sql.Date(System.currentTimeMillis());
        this.altPor = "Admin";
        try {
            con = getConnection();
            query = con.prepareStatement("update itens set "
                + "descricao = ?, prodComp = ?, finalInter = ?, realFant = ?, "
                + "calcMRP = ?, unidade = ?, custo = ?, qtdeLote = ?, leadTime = ?,
"
                + "estSeguranca = ?, dataAlt = ?, altPor = ? where codigo = ?");
            query.setString(1, descricao);
            query.setString(2, prodComp);
            query.setString(3, finalInter);
            query.setString(4, realFant);
            query.setBoolean(5, calcMRP);
            query.setString(6, unidade);
            query.setDouble(7, custo);
            query.setDouble(8, qtdeLote);
            query.setDouble(9, leadTime);
            query.setDouble(10, estSeguranca);
            query.setString(11, new java.text.SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(this.dataAlt));
            query.setString(12, this.altPor);
            query.setString(13, this.codigo);
            query.executeUpdate();
            System.out.println("update itens set ..... where codigo =
"+this.codigo);
        }
        catch (Exception e) {
            throw new EJBException("Não foi possível salvar o item "+this.codigo,
e);
        }
        finally {
            try { if (query != null) query.close(); }
            catch (Exception e) {}
            try { if (con != null) con.close(); }
            catch (Exception e) {}
        }
    }

    public void ejbRemove() throws RemoveException {
        ItemBMPPK pk = (ItemBMPPK) ctx.getPrimaryKey();
        this.codigo = pk.codigo;
        Connection con = null;
        PreparedStatement query = null;
        try {
            con = getConnection();
            query = con.prepareStatement("delete from itens where codigo = ?");
            query.setString(1, this.codigo);
            if (query.executeUpdate() == 0) {
                throw new RemoveException("Não foi possível apagar o item
"+this.codigo);
            }
        }
        catch (Exception e) {
            throw new EJBException(e.toString());
        }
        finally {
            try { if (query != null) query.close(); }
            catch (Exception e) {}
            try { if (con != null) con.close(); }
            catch (Exception e) {}
        }
    }
```

```
    public ItemBMPPK ejbFindByPrimaryKey(ItemBMPPK pKey) throws FinderException {
        Connection con = null;
        PreparedStatement query = null;
        try {
            con = getConnection();
            query = con.prepareStatement("select codigo from itens where codigo =
?");
            query.setString(1, pKey.codigo);
            ResultSet rs = query.executeQuery();
            rs.next();
            return pKey;
        }
        catch (Exception e) {
            throw new FinderException(e.toString());
        }
        finally {
            try { if (query != null) query.close(); }
            catch (Exception e) {}
            try { if (con != null) con.close(); }
            catch (Exception e) {}
        }
    }

    public Collection ejbFindByDescricao(String descricao) throws FinderException {
        Connection con = null;
        PreparedStatement query = null;
        Vector v = new Vector();
        try {
            System.out.println("chamando ejbFindByDescricao("+descricao+")");
            con = getConnection();
            String qString = "select codigo from itens where descricao like
'%"+descricao+"%' order by descricao";
            query = con.prepareStatement(qString);
            //query.setString(1, descricao);
            ResultSet rs = query.executeQuery();
            while (rs.next()) {
                String codigo = rs.getString("codigo");
                v.addElement(new ItemBMPPK(codigo));
            }
            return v;
        }
        catch (Exception e) {
            throw new FinderException(e.toString());
        }
        finally {
            try { if (query != null) query.close(); }
            catch (Exception e) {}
            try { if (con != null) con.close(); }
            catch (Exception e) {}
        }
    }
}
```

codigo 1. ItemBMPBean.java

Percebe-se pelo código que todos os métodos são precedidos por comentários. Esses comentários, além de permitirem a geração automática de documentação (via *javadoc*), são necessários para a ferramenta Xdoclet (ver item 4.5.1).

Através do parâmetro "`@ejb:pk-field`", por exemplo, o XdocLet gera a chave primária (ver codigo 4). Através do parâmetro "`@ejb:interface-method view-type="remote"`" o XdocLet incluí o método na interface remota (ver codigo 2) e assim por diante[2].

Assim, com o uso do XdocLet, o arquivo "??Bean" é o único efetivamente escrito pelo programador. Os arquivos auxiliares (ver abaixo) foram todos gerados. No arquivo ItemBMPBean.java são definidos então todos os atributos e métodos do EJB. Como se trata de um Entity Bean do tipo BMP (*Bean Managed Persistence*), todos os métodos para manutenção da persistência no banco são também definidos nesse arquivo. Esses métodos são obrigatórios e dizem ao *container* como ele deve agir quando quiser criar (**ejbCreate()**), salvar (**ejbStore()**), recuperar (**ejbLoad()**) ou remover (**ejbRemove()**) um determinado objeto do banco.

Além disso são também obrigatórios os métodos **ejbPassivate()** e **ejbActivate()** que dão ao *container* eventuais recomendações especiais quando determinado objeto for tirado de atividade (para economizar recursos do sistema) ou recolocado em atividade respectivamente. Por fim são também obrigatórios os métodos **set/unsetEntityContext()** (associa o EJB a um contexto para que este possa prover informações do sistema) e **ejbPostCreate()** (que é chamado após a criação de um EJB).

A seguir são apresentados os outros arquivos que compõe o EJB e que, no caso, são gerados pelo Xdoclet.

```
/*
 * Generated file - Do not edit!
 */
package MRP;

import java.lang.*;
import java.sql.*;
import java.util.*;
import javax.ejb.*;
import javax.naming.*;
import MRP.ItemBMPPK;

/**
 * Remote interface for MRP/ItemBMP.
 * @author XDOCLET 1.1.2
 * @xdoclet-generated at 19/06/2003 18:45:33
 */
public interface ItemBMP
   extends javax.ejb.EJBObject
{
   /**
    * Atualiza o EJB
    */
   public void atualiza(  ) throws java.rmi.RemoteException;

   /**
```

[2] Para documentação completa ver http://www.xdoclet.com/

```
    * Retorna Alterado por?
    * @return altPor
    */
   public java.lang.String getAltPor(  ) throws java.rmi.RemoteException;

   /**
    * Retorna calcMRP (true se for calculado por MRP)
    * @return calcMRP
    */
   public boolean getCalcMRP(  ) throws java.rmi.RemoteException;

   /**
    * Retorna o código
    * @return Código do item
    */
   public java.lang.String getCodigo(  ) throws java.rmi.RemoteException;

   /**
    * Retorna o custo
    * @return custo
    */
   public double getCusto(  ) throws java.rmi.RemoteException;

   /**
    * Retorna a data de alteração
    * @return dataAlt
    */
   public java.sql.Date getDataAlt(  ) throws java.rmi.RemoteException;

   /**
    * Retorna a descrição
    * @return Descrição
    */
   public java.lang.String getDescricao(  ) throws java.rmi.RemoteException;

   /**
    * Retorna o Estoque de Segurança
    * @return estSeguranca
    */
   public double getEstSeguranca(  ) throws java.rmi.RemoteException;

   /**
    * Retorna finalInter (final-"F" ou Intermediário "I")
    * @return finalInter
    */
   public java.lang.String getFinalInter(  ) throws java.rmi.RemoteException;

   /**
    * Retorna o LLC (Low Level Code)
    * @return LLC
    */
   public int getLLC(  ) throws java.rmi.RemoteException;

   /**
    * Retorna o Lead Time
    * @return leadTime
    */
   public double getLeadTime(  ) throws java.rmi.RemoteException;

   /**
    * Retorna prodComp (produzido-"P" ou comprado "C")
    * @return prodComp
    */
   public java.lang.String getProdComp(  ) throws java.rmi.RemoteException;

   /**
```

```
     * Retorna o lote
     * @return qtdeLote
     */
    public double getQtdeLote(  ) throws java.rmi.RemoteException;

    /**
     * Retorna realFant (Real-"R" ou Fantasia "F")
     * @return realFant
     */
    public java.lang.String getRealFant(  ) throws java.rmi.RemoteException;

    /**
     * Retorna a unidade (m, KG, ml etc.)
     * @return unidade
     */
    public java.lang.String getUnidade(  ) throws java.rmi.RemoteException;

    /**
     * Seta realFant
     * @param pRealFant
     */
    public void setCalcMRP( boolean pCalcMRP ) throws java.rmi.RemoteException;

    /**
     * Seta o código
     * @param pCodigo The id of this TestEntity. Is set at creation time.
     */
    public void setCodigo( java.lang.String pCodigo ) throws
java.rmi.RemoteException;

    /**
     * Seta o custo
     * @param pCusto
     */
    public void setCusto( double pCusto ) throws java.rmi.RemoteException;

    /**
     * Seta a Descrição
     * @param pDescricao
     */
    public void setDescricao( java.lang.String pDescricao ) throws
java.rmi.RemoteException;

    /**
     * Seta estSeguranca
     * @param pEstSeguranca
     */
    public void setEstSeguranca( double pEstSeguranca ) throws
java.rmi.RemoteException;

    /**
     * Seta finalInter
     * @param pfinalInter
     */
    public void setFinalInter( java.lang.String pfinalInter ) throws
java.rmi.RemoteException;

    /**
     * Seta leadTime
     * @param pLeadTime
     */
    public void setLeadTime( double pLeadTime ) throws java.rmi.RemoteException;

    /**
     * Seta prodComp
     * @param pProdComp
```

```
    */
   public void setProdComp( java.lang.String pProdComp ) throws
java.rmi.RemoteException;

   /**
    * Seta qtdeLote
    * @param pQtdeLote
    */
   public void setQtdeLote( double pQtdeLote ) throws java.rmi.RemoteException;

   /**
    * Seta realFant
    * @param pRealFant
    */
   public void setRealFant( java.lang.String pRealFant ) throws
java.rmi.RemoteException;

   /**
    * Seta a unidade
    * @param pUnidade
    */
   public void setUnidade( java.lang.String pUnidade ) throws
java.rmi.RemoteException;

}
```

codigo 2. ItemBMP.java

Essa constitui a interface de fato do EJB e traz todos os métodos que ele pode realizar. Percebe-se que a classe herda diretamente de **EJBObject**.

Como padrão de modelagem, define-se que as variáveis só podem ser consultadas ou alteradas por meio dos métodos **get** e **set** respectivamente.

```
/*
 * Generated file - Do not edit!
 */
package MRP;

import java.lang.*;
import java.sql.*;
import java.util.*;
import javax.ejb.*;
import javax.naming.*;
import MRP.ItemBMPPK;

/**
 * Home interface for MRP/ItemBMP. Lookup using {1}
 * @author XDOCLET 1.1.2
 * @xdoclet-generated at 19/06/2003 18:45:31
 */
public interface ItemBMPHome
   extends javax.ejb.EJBHome
{
   public static final String COMP_NAME="java:comp/env/ejb/MRP/ItemBMP";
   public static final String JNDI_NAME="ejb/MRP/ItemBMP";

   /**
    * Cria um item e insere no banco de dados
    * @param codigo, descricao, prodComp, finalInter, realFant,
       calcMRP, unidade, custo, qtdeLote, leadTime, estSeguranca
    * @throws CreateException
    * @return chave primária
    */
```

```
   public MRP.ItemBMP create(java.lang.String codigo,java.lang.String
descricao,java.lang.String prodComp,java.lang.String finalInter,java.lang.String
realFant,boolean calcMRP,java.lang.String unidade,double custo,double
qtdeLote,double leadTime,double estSeguranca) throws
java.rmi.RemoteException,javax.ejb.CreateException;

   public java.util.Collection findByDescricao(java.lang.String descricao)
      throws java.rmi.RemoteException,javax.ejb.FinderException;

   public MRP.ItemBMP findByPrimaryKey(MRP.ItemBMPPK pKey)
      throws java.rmi.RemoteException,javax.ejb.FinderException;

}
```

codigo 3. ItemBMPHome.java

Trata-se da interface Home (que herda de **EJBHome**). Como discutido no item 3.2.2 essa classe serve de "fábrica" para os objetos EJB e pode conter métodos de negócio que dizem respeito a um conjunto de EJB (de itens de estoque nesse caso).

Nessa classe são definidos os métodos de pesquisa (que começam com **findBy**). Nota-se que o Xdoclet atribui esses métodos à interface *Home* automaticamente.

Nesse EJB em particular não temos exemplo de um método de negócio realizado pela interface *Home*. Isso pode ser feito escrevendo-se o método no arquivo "??Bean" (que deve ser iniciado necessariamente por ejbHome) e automatizado no Xdoclet pelo uso de um parâmetro "`@ejb:home-method view-type="remote"`". Embora seja iniciado por "ejbHome" no arquivo original, o método na interface *Home* será criado sem esse radical e com a primeira letra subsequente convertida para minúscula (de forma similar ao que ocorre com os métodos "findBy")[3].

Por fim temos a clase ItemBMPPK que é gerada para cada *Entity Bean* e que representa um Objeto de forma única.

```
/*
 * Generated file - Do not edit!
 */
package MRP;

import java.lang.*;
import java.sql.*;
import java.util.*;
import javax.ejb.*;
import javax.naming.*;

/**
 * Primary key for MRP/ItemBMP.
 * @author XDOCLET 1.1.2
 * @xdoclet-generated at 19/06/2003 18:45:34
 */
public class ItemBMPPK
   extends java.lang.Object
   implements java.io.Serializable
{
```

[3] Ver código em anexo para exemplos dessa utilização.

```
static final long serialVersionUID = 673561001582958179 4L;
transient private int _hashCode = Integer.MIN_VALUE;
transient private String value = null;

public java.lang.String codigo;

public ItemBMPPK()
{
}

public ItemBMPPK( java.lang.String codigo )
{
   this.codigo = codigo;
}

public java.lang.String getCodigo()
{
   return codigo;
}

public void setCodigo(java.lang.String codigo)
{
   this.codigo = codigo;
}

public int hashCode()
{
   if( _hashCode == Integer.MIN_VALUE )
   {
      _hashCode += this.codigo.hashCode();
   }

   return _hashCode;
}

public boolean equals(Object obj)
{
   if( !(obj instanceof MRP.ItemBMPPK) )
      return false;

   MRP.ItemBMPPK pk = (MRP.ItemBMPPK)obj;
   boolean eq = true;

   if( obj == null )
   {
      eq = false;
   }
   else
   {
      if( this.codigo == null && ((MRP.ItemBMPPK)obj).getCodigo() == null )
      {
         eq = true;
      }
      else
      {
         if( this.codigo == null || ((MRP.ItemBMPPK)obj).getCodigo() == null )
         {
            eq = false;
         }
         else
         {
            eq = eq && this.codigo.equals( pk.codigo );
         }
      }
   }
```

```
        return eq;
    }

    public String toString()
    {
        if( value == null )
        {
            value = "[.";
            value += this.codigo+".";
            value += "]";
        }

        return value;
    }
}
```

codigo 4. ItemBMPPK.java

Além desses existem outros arquivos marginais que são criados para uso do *container*. Entre eles o principal é o *deployment descriptor* (ejb-jar.xml). Trata-se de um arquivo escrito em XML (Extended Markup Language) que traz informações sobre todos os EJBs existentes no sistema e é utilizado para organização do *container*. No nosso caso esse arquivo também é gerado pelo Xdoclet a partir de parâmetros adicionados no início dos arquivos "??Bean.java". Ele está reproduzido a seguir para todos os EJBs existentes no sistema.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb-jar >

   <description>No Description.</description>
   <display-name>Generated by XDoclet</display-name>

   <enterprise-beans>

      <!-- Session Beans -->
      <session >
         <description><![CDATA[Manipulacao da BOM - Bill of Materials
(Stateful)]]></description>
         <display-name>ManipBOM</display-name>

         <ejb-name>MRP/ManipBOM</ejb-name>

         <home>MRP.ManipBOMHome</home>
         <remote>MRP.ManipBOM</remote>
         <ejb-class>MRP.ManipBOMBean</ejb-class>
         <session-type>Stateful</session-type>
         <transaction-type>Container</transaction-type>

      </session>

      <session >
         <description><![CDATA[Manipulacao da BOM - Bill of Materials
(Stateful)]]></description>
         <display-name>ManipCap</display-name>

         <ejb-name>MRP/ManipCap</ejb-name>

         <home>MRP.ManipCapHome</home>
```

```
         <remote>MRP.ManipCap</remote>
         <ejb-class>MRP.ManipCapBean</ejb-class>
         <session-type>Stateful</session-type>
         <transaction-type>Container</transaction-type>

      </session>

      <session >
         <description><![CDATA[Manipulacao de Itens (Stateful)]]></description>
         <display-name>ManipItens</display-name>

         <ejb-name>MRP/ManipItens</ejb-name>

         <home>MRP.ManipItensHome</home>
         <remote>MRP.ManipItens</remote>
         <ejb-class>MRP.ManipItensBean</ejb-class>
         <session-type>Stateful</session-type>
         <transaction-type>Container</transaction-type>

      </session>

      <session >
         <description><![CDATA[Execucao do calculo de MRP
(Stateful)]]></description>
         <display-name>MRPExec</display-name>

         <ejb-name>MRP/MRPExec</ejb-name>

         <home>MRP.MRPExecHome</home>
         <remote>MRP.MRPExec</remote>
         <ejb-class>MRP.MRPExecBean</ejb-class>
         <session-type>Stateful</session-type>
         <transaction-type>Container</transaction-type>

      </session>

      <session >
         <description><![CDATA[Exemplo "Count" (Stateful)]]></description>
         <display-name>Count</display-name>

         <ejb-name>testes/Count</ejb-name>

         <home>testes.CountHome</home>
         <remote>testes.Count</remote>
         <ejb-class>testes.CountBean</ejb-class>
         <session-type>Stateful</session-type>
         <transaction-type>Container</transaction-type>

      </session>

      <session >
         <description><![CDATA[Exemplo Session Bean "Hello"]]></description>
         <display-name>Hello World</display-name>

         <ejb-name>testes/Hello</ejb-name>

         <home>testes.HelloHome</home>
         <remote>testes.Hello</remote>
         <ejb-class>testes.HelloBean</ejb-class>
         <session-type>Stateless</session-type>
         <transaction-type>Container</transaction-type>

      </session>

     <!--
       To add session beans that you have deployment descriptor info for, add
```

```
        a file to your merge directory called session-beans.xml that contains
        the <session></session> markup for those beans.
      -->

       <!-- Entity Beans -->
       <entity >
          <description><![CDATA[Entity Bean (BMP) para itens de
capacidade]]></description>
          <display-name>Entity Bean (BMP) para itens de capacidade</display-name>

          <ejb-name>MRP/Capacidade</ejb-name>

          <home>MRP.CapacidadeHome</home>
          <remote>MRP.Capacidade</remote>

          <ejb-class>MRP.CapacidadeBMP</ejb-class>
          <persistence-type>Bean</persistence-type>
          <prim-key-class>MRP.CapacidadePK</prim-key-class>
          <reentrant>False</reentrant>

       </entity>

       <entity >
          <description><![CDATA[Entity Bean (BMP) para itens de
estoque]]></description>
          <display-name>Entity Bean (BMP) para itens de estoque</display-name>

          <ejb-name>MRP/Estoque</ejb-name>

          <home>MRP.EstoqueHome</home>
          <remote>MRP.Estoque</remote>

          <ejb-class>MRP.EstoqueBMP</ejb-class>
          <persistence-type>Bean</persistence-type>
          <prim-key-class>MRP.EstoquePK</prim-key-class>
          <reentrant>False</reentrant>

       </entity>

       <entity >
          <description><![CDATA[Entity Bean (BMP) para itens de
estoque]]></description>
          <display-name>Entity Bean (BMP) para itens de estoque</display-name>

          <ejb-name>MRP/ItemBMP</ejb-name>

          <home>MRP.ItemBMPHome</home>
          <remote>MRP.ItemBMP</remote>

          <ejb-class>MRP.ItemBMPBMP</ejb-class>
          <persistence-type>Bean</persistence-type>
          <prim-key-class>MRP.ItemBMPPK</prim-key-class>
          <reentrant>False</reentrant>

       </entity>

       <entity >
          <description><![CDATA[Entity Bean (BMP) para itens da lista de materiais
BOM]]></description>
          <display-name>Entity Bean (BMP) para itens da lista de materiais
BOM</display-name>

          <ejb-name>MRP/ItemBOM</ejb-name>

          <home>MRP.ItemBOMHome</home>
          <remote>MRP.ItemBOM</remote>
```

```
            <ejb-class>MRP.ItemBOMBMP</ejb-class>
            <persistence-type>Bean</persistence-type>
            <prim-key-class>MRP.ItemBOMPK</prim-key-class>
            <reentrant>False</reentrant>
            <primkey-field>codItemBOM</primkey-field>

         </entity>

         <entity >
            <description><![CDATA[Entity Bean (BMP) para processos]]></description>
            <display-name>Entity Bean (BMP) para processos</display-name>

            <ejb-name>MRP/Processo</ejb-name>

            <home>MRP.ProcessoHome</home>
            <remote>MRP.Processo</remote>

            <ejb-class>MRP.ProcessoBMP</ejb-class>
            <persistence-type>Bean</persistence-type>
            <prim-key-class>MRP.ProcessoPK</prim-key-class>
            <reentrant>False</reentrant>

         </entity>

         <entity >
            <description><![CDATA[Entity Bean (BMP) para itens de
recurso]]></description>
            <display-name>Entity Bean (BMP) para itens de recurso</display-name>

            <ejb-name>MRP/Recurso</ejb-name>

            <home>MRP.RecursoHome</home>
            <remote>MRP.Recurso</remote>

            <ejb-class>MRP.RecursoBMP</ejb-class>
            <persistence-type>Bean</persistence-type>
            <prim-key-class>MRP.RecursoPK</prim-key-class>
            <reentrant>False</reentrant>

         </entity>

      <!--
         To add entity beans that you have deployment descriptor info for, add
         a file to your merge directory called entity-beans.xml that contains
         the <entity></entity> markup for those beans.
      -->

         <!-- Message Driven Beans -->
      <!--
         To add message driven beans that you have deployment descriptor info for,
add
         a file to your merge directory called message-driven-beans.xml that contains
         the <message-driven></message-driven> markup for those beans.
      -->

   </enterprise-beans>

   <!-- Relationships -->

   <!-- Assembly Descriptor -->
   <assembly-descriptor >

   <!-- finder permissions -->

   <!-- transactions -->
```

```
    <!-- finder transactions -->
    </assembly-descriptor>

</ejb-jar>
```

codigo 5. ejb-jar.xml

Esses arquivos formam então a camada do servidor para nosso EJB de itens de estoque. Resta então um cliente que utilize todas essas classes. Para isso temos uma página JSP construída para realizar todas as operações relacionadas aos itens. Segue o código para essa página.

```
<%@page contentType="text/html"
    import="javax.naming.*,testes.*,MRP.*,java.util.*"
%>
<html>
<head><title>Manipulação de itens</title>
<link  rel=STYLESHEET href="styles.css" type="text/css">
</head>
<body>
<%
if (request.getParameterValues("acao") == null) {
%>  <BR>
    <A href='itensManip.jsp?acao=inserir'><B><font color='darkblue' size='2'>Criar
novo item</FONT></B></A><BR>
    <A href='itensManip.jsp?acao=busca'><B><font color='darkblue'
size='2'>Pesquisar itens</FONT></B></A><BR>
<%
}
else {
    String acao = request.getParameterValues("acao")[0];
    if (acao.compareTo("inserir") == 0) {
    try {
        if (request.getParameterValues("codigo") != null) {
            String codigo = request.getParameterValues("codigo")[0]; // PK
            String descricao = request.getParameterValues("descricao")[0];
            String prodComp = request.getParameterValues("prodComp")[0];
            String finalInter = request.getParameterValues("finalInter")[0];
            String realFant = request.getParameterValues("realFant")[0];
            boolean calcMRP =
request.getParameterValues("calcMRP")[0].compareTo("Sim") == 0 ? true : false;
            String unidade = request.getParameterValues("unidade")[0];
            double custo =
Double.parseDouble(request.getParameterValues("custo")[0]);
            double qtdeLote =
Double.parseDouble(request.getParameterValues("qtdeLote")[0]);
            double leadTime =
Double.parseDouble(request.getParameterValues("leadTime")[0]);
            //int LLC = request.getParameterValues("LLC")[1];
            double estSeguranca =
Double.parseDouble(request.getParameterValues("estSeguranca")[0]);

            Context ctx = new InitialContext();
            ManipItensHome manipHome = (ManipItensHome) ctx.lookup(
                "java:comp/env/ejb/MRP/ManipItens"
            );
            ManipItens manip = manipHome.create();

            ItemBMP item = manip.criar(codigo, descricao, prodComp, finalInter,
realFant, calcMRP, unidade, custo, qtdeLote, leadTime, estSeguranca);

%>
<FORM name='manip' action='itensManip.jsp' method=post>
```

```
<INPUT type=hidden name='acao' value=''>
<TABLE border='0' cellpadding='0' cellspacing='0'>
<TR><TD colspan='2' align='center'> </TR>
<TR><TD colspan='2' align='center'><H2 align='center'>Item <%=codigo%> criado com
sucesso</H2></TR>
<TR>
    <TD width='400' valign='top'>
        <TABLE border='0' cellpadding='0' cellspacing='5'>
            <TR><TD width='200'>Código</TD>
                <TD><INPUT type=text name='codigo' value='<%=codigo%>'></TD>
            </TR>
            <TR><TD width='200'>Descrição</TD>
                <TD >
                    <INPUT type=text name='descricao' value='<%=descricao%>'></TD>
            </TR>
            <TR><TD width='200' >Unidade</TD>
                <TD ><SELECT name='unidade'>
                    <OPTION value=''>Escolha uma unidade de medida
                    <OPTION value='g' <%=(unidade.compareTo("g")==0) ? "selected" :
""%>>g
                    <OPTION value='ml' <%=(unidade.compareTo("ml")==0) ? "selected"
: ""%>>ml
                    <OPTION value='cm' <%=(unidade.compareTo("cm")==0) ? "selected"
: ""%>>cm
                    <OPTION value='unidades' <%=(unidade.compareTo("unidades")==0)
? "selected" : ""%>>unidades
                    </SELECT></TD>
            </TR>
            <TR><TD width='200'>Custo</TD>
                <TD><INPUT type=text name='custo' value='<%=custo%>'></TD>
            </TR>
            <TR><TD width='200'>Quantidade do lote</TD>
                <TD><INPUT type=text name='qtdeLote' value='<%=qtdeLote%>'></TD>
            </TR>
            <TR><TD width='200'>Lead time</TD>
                <TD><INPUT type=text name='leadTime' value='<%=leadTime%>'></TD>
            </TR>
            <TR><TD width='200' >Estoque de Segurança</TD>
                <TD >
                    <INPUT type=text name='estSeguranca'
value='<%=estSeguranca%>'></TD>
            </TR>
        </TABLE>
    </TD>
    <TD align='top'>
        <TABLE border='1' cellpadding='5' cellspacing='0'>
            <TR><TD width='200'><INPUT type=radio name='prodComp' value='Produzido'
<%=(prodComp.compareTo("Produzido")==0) ? "checked" : ""%>>Produzido<BR>
                    <INPUT type=radio name='prodComp' value='Comprado'
<%=(prodComp.compareTo("Comprado")==0) ? "checked" : ""%>>Comprado</TD>
            </TR>
            <TR><TD width='200'><INPUT type=radio name='finalInter' value='Final'
<%=(finalInter.compareTo("Final")==0) ? "checked" : ""%>>Final<BR>
                    <INPUT type=radio name='finalInter' value='Intermediario'
<%=(finalInter.compareTo("Intermediario")==0) ? "checked" : ""%>>Intermediario</TD>
            </TR>
            <TR><TD><INPUT type=radio name='realFant' value='Real'
<%=(realFant.compareTo("Real")==0) ? "checked" : ""%>>Real<BR>
                    <INPUT type=radio name='realFant' value='Fantasma'
<%=(realFant.compareTo("Fantasma")==0) ? "checked" : ""%>>Fantasma</TD>
            </TR>
            <TR><TD>Calculado com MRP?<BR>
                    <INPUT type=radio name='calcMRP' value='Sim' <%=(calcMRP) ?
"checked" : ""%>>Sim<BR>
                    <INPUT type=radio name='calcMRP' value='Nao' <%=(calcMRP) ? ""
: "checked"%>>Não</TD>
```

```
            </TR>
        </TABLE>
    </TD>
</TR></TABLE>
    <TABLE border='0' cellpadding='5' cellspacing='0'>
        <TR><TD><INPUT type=submit name='alterar' value='Alterar'
onclick='document.manip.acao.value="alterar";'>
            </TD>
            <TD><INPUT type=submit name='remover' value='Remover'
onclick='document.manip.acao.value="remover";'>
            </TD>
            <TD><INPUT type=button name='processos' value='Processos'
onclick='document.location.href="processosManip.jsp?acao=busca&codItem=<%=codigo%>&
codRecurso=";'>
            </TD>
        </TR>
    </TABLE>
</FORM>
<%      }
        else
%>
<FORM action= 'itensManip.jsp?acao=inserir' method=post>
<!--<FORM action= 'echo.jsp' method=post>-->
<TABLE border='0' cellpadding='0' cellspacing='0'>
<TR><TD colspan='2' align='center'> </TR>
<TR><TD colspan='2' align='center'><H2 align='center'>Criar novo item:</H2></TR>
<TR>
    <TD width='400' valign='top'>
        <TABLE border='0' cellpadding='0' cellspacing='5'>
            <TR><TD width='200'>Código</TD>
                <TD><INPUT type=text name='codigo' value=''></TD>
            </TR>
            <TR><TD width='200'>Descrição</TD>
                <TD >
                    <INPUT type=text name='descricao' value=''></TD>
            </TR>
            <TR><TD width='200' >Unidade</TD>
                <TD ><SELECT name='unidade'>
                    <OPTION value='' selected>Escolha uma unidade de medida
                    <OPTION value='g'>g
                    <OPTION value='ml'>ml
                    <OPTION value='cm'>cm
                    <OPTION value='unidades'>unidades
                    </SELECT></TD>
            </TR>
            <TR><TD width='200'>Custo</TD>
                <TD><INPUT type=text name='custo' value='0.00'></TD>
            </TR>
            <TR><TD width='200'>Quantidade do lote</TD>
                <TD><INPUT type=text name='qtdeLote'></TD>
            </TR>
            <TR><TD width='200'>Lead time</TD>
                <TD><INPUT type=text name='leadTime'></TD>
            </TR>
            <TR><TD width='200' >Estoque de Segurança</TD>
                <TD >
                    <INPUT type=text name='estSeguranca' value=''></TD>
            </TR>
        </TABLE>
    </TD>
    <TD align='top'>
        <TABLE border='1' cellpadding='5' cellspacing='0'>
            <TR><TD width='200'><INPUT type=radio name='prodComp' value='Produzido'
checked>Produzido<BR>
                    <INPUT type=radio name='prodComp'
value='Comprado'>Comprado</TD>
```

```
            </TR>
            <TR><TD width='200'><INPUT type=radio name='finalInter'
value='Final'>Final<BR>
                    <INPUT type=radio name='finalInter' value='Intermediario'
checked>Intermediario</TD>
            </TR>
            <TR><TD><INPUT type=radio name='realFant' value='Real' checked>Real<BR>
                    <INPUT type=radio name='realFant'
value='Fantasma'>Fantasma</TD>
            </TR>
            <TR><TD>Calculado com MRP?<BR>
                    <INPUT type=radio name='calcMRP' value='Sim' checked>Sim<BR>
                    <INPUT type=radio name='calcMRP' value='Nao'>Não</TD>
            </TR>
        </TABLE>
        <TABLE border='0' cellpadding='5' cellspacing='0'>
            <TR><TD width='200' align='right'><INPUT type=submit name='criar'
value='Criar'></TD>
            </TR>
        </TABLE>
    </TD>
</TR></TABLE>
        </FORM>
<%
        }    //Primeiro "try"
        catch (Exception e) {%>Erro<BR><%out.println(e.toString());
             e.printStackTrace();
        }
    } // "if (acao.compareTo("inserir") == 0)"
    else if (acao.compareTo("busca") == 0) {
    try {
        if (request.getParameterValues("codigo") != null &&
request.getParameterValues("codigo")[0].compareTo("")!=0) {
            String codigo = request.getParameterValues("codigo")[0];
            Context ctx = new InitialContext();
            ManipItensHome manipHome = (ManipItensHome) ctx.lookup(
                 "java:comp/env/ejb/MRP/ManipItens"
            );
            ManipItens manip = manipHome.create();
            ItemBMP item = manip.findByPrimaryKey(codigo);

            String descricao = item.getDescricao();
            String prodComp = item.getProdComp();
            String finalInter = item.getFinalInter();
            String realFant = item.getRealFant();
            boolean calcMRP = item.getCalcMRP();
            String unidade = item.getUnidade();
            double custo = item.getCusto();
            double qtdeLote = item.getQtdeLote();
            double leadTime = item.getLeadTime();
            double estSeguranca = item.getEstSeguranca();
%>
<!--<FORM action= 'echo.jsp' method=post>-->
<!-- action= 'itensManip.jsp?acao=alterar' -->
<FORM name='manip' action='itensManip.jsp' method=post>
<INPUT type=hidden name='acao' value=''>
<TABLE border='0' cellpadding='0' cellspacing='0'>
<TR><TD colspan='2' align='center'> </TR>
<TR><TD colspan='2' align='center'><H2 align='center'>Item <%=codigo%>
encontrado:</H2></TR>
<TR>
    <TD width='400' valign='top'>
        <TABLE border='0' cellpadding='0' cellspacing='5'>
            <TR><TD width='200'>Código</TD>
                <TD><INPUT type=text name='codigo' value='<%=codigo%>'></TD>
            </TR>
```

```
            <TR><TD width='200'>Descrição</TD>
                <TD >
                    <INPUT type=text name='descricao' value='<%=descricao%>'></TD>
            </TR>
            <TR><TD width='200' >Unidade</TD>
                <TD ><SELECT name='unidade'>
                    <OPTION value=''>Escolha uma unidade de medida
                    <OPTION value='g' <%=(unidade.compareTo("g")==0) ? "selected" :
""%>>g
                    <OPTION value='ml' <%=(unidade.compareTo("ml")==0) ? "selected"
: ""%>>ml
                    <OPTION value='cm' <%=(unidade.compareTo("cm")==0) ? "selected"
: ""%>>cm
                    <OPTION value='unidades' <%=(unidade.compareTo("unidades")==0)
? "selected" : ""%>>unidades
                    </SELECT></TD>
            </TR>
            <TR><TD width='200'>Custo</TD>
                <TD><INPUT type=text name='custo' value='<%=custo%>'></TD>
            </TR>
            <TR><TD width='200'>Quantidade do lote</TD>
                <TD><INPUT type=text name='qtdeLote' value='<%=qtdeLote%>'></TD>
            </TR>
            <TR><TD width='200'>Lead time</TD>
                <TD><INPUT type=text name='leadTime' value='<%=leadTime%>'></TD>
            </TR>
            <TR><TD width='200' >Estoque de Segurança</TD>
                <TD >
                    <INPUT type=text name='estSeguranca'
value='<%=estSeguranca%>'></TD>
            </TR>
        </TABLE>
    </TD>
    <TD align='top'>
        <TABLE border='1' cellpadding='5' cellspacing='0'>
            <TR><TD width='200'><INPUT type=radio name='prodComp' value='Produzido'
<%=(prodComp.compareTo("Produzido")==0) ? "checked" : ""%>>Produzido<BR>
                    <INPUT type=radio name='prodComp' value='Comprado'
<%=(prodComp.compareTo("Comprado")==0) ? "checked" : ""%>>Comprado</TD>
            </TR>
            <TR><TD width='200'><INPUT type=radio name='finalInter' value='Final'
<%=(finalInter.compareTo("Final")==0) ? "checked" : ""%>>Final<BR>
                    <INPUT type=radio name='finalInter' value='Intermediario'
<%=(finalInter.compareTo("Intermediario")==0) ? "checked" : ""%>>Intermediario</TD>
            </TR>
            <TR><TD><INPUT type=radio name='realFant' value='Real'
<%=(realFant.compareTo("Real")==0) ? "checked" : ""%>>Real<BR>
                    <INPUT type=radio name='realFant' value='Fantasma'
<%=(realFant.compareTo("Fantasma")==0) ? "checked" : ""%>>Fantasma</TD>
            </TR>
            <TR><TD>Calculado com MRP?<BR>
                    <INPUT type=radio name='calcMRP' value='Sim' <%=(calcMRP) ?
"checked" : ""%>>Sim<BR>
                    <INPUT type=radio name='calcMRP' value='Nao' <%=(calcMRP) ? ""
: "checked"%>>Não</TD>
            </TR>
        </TABLE>
    </TD>
</TR></TABLE>
    <TABLE border='0' cellpadding='5' cellspacing='0'>
        <TR><TD><INPUT type=submit name='alterar' value='Alterar'
onclick='document.manip.acao.value="alterar";'>
            </TD>
            <TD><INPUT type=submit name='remover' value='Remover'
onclick='document.manip.acao.value="remover";'>
            </TD>
```

```
            <TD><INPUT type=button name='processos' value='Processos'
onclick='document.location.href="processosManip.jsp?acao=busca&codItem=<%=codigo%>&
codRecurso=";'>
            </TD>
        </TR>
    </TABLE>
</FORM>
<%
        }
        else if (request.getParameterValues("descricao") != null &&
request.getParameterValues("descricao")[0].compareTo("")!=0) {
            String descricao = request.getParameterValues("descricao")[0];
            Context ctx = new InitialContext();
            ManipItensHome manipHome = (ManipItensHome) ctx.lookup(
                 "java:comp/env/ejb/MRP/ManipItens"
            );
            ManipItens manip = manipHome.create();
            ItemBMP item = null;
            Iterator i = manip.findByDescricao(descricao);
%>
<TABLE border='0' cellpadding='1' cellspacing='1'>
<TR><TD align='center'> </TR>
<TR><TD align='center'><H2 align='center'>Resultado da pesquisa:</H2></TR>
<%            while (i.hasNext()) {
                item = (ItemBMP) i.next();
                out.println("<TR><TD><A
href='itensManip.jsp?acao=busca&codigo="+item.getCodigo()+
                    "'>Item: "+item.getDescricao()+", codigo:
"+item.getCodigo()+"</A></TD></TR>");
              }
              out.println("</TABLE>");
        }
        else
%>
        <FORM action='itensManip.jsp?acao=busca' method=post>
        <TABLE border='0' cellpadding='0' cellspacing='0'>
        <TR><TD align='center'><H2 align='center'>Pesquisa de item(ns)</H2></TR>
        <TR><TD width='200'>Código</TD>
            <TD><INPUT type=text name='codigo'></TD>
        </TR>
        <TR><TD width='200'>Descrição</TD>
            <TD><INPUT type=text name='descricao' value=''></TD>
        </TR>
        <TR><TD width='200'></TD>
            <TD><INPUT type=submit name='buscar' value='Buscar'></TD>
        </TR>
        </TABLE>
        </FORM>
<%
        }    //"try"
        catch (Exception e) {%><%
            out.println("<br>O item não foi encontrado");
            e.printStackTrace();
        }
    }
    else if (acao.compareTo("alterar") == 0) {
    try {
        String codigo = request.getParameterValues("codigo")[0]; // PK
        String descricao = request.getParameterValues("descricao")[0];
        String prodComp = request.getParameterValues("prodComp")[0];
        String finalInter = request.getParameterValues("finalInter")[0];
        String realFant = request.getParameterValues("realFant")[0];
        boolean calcMRP = request.getParameterValues("calcMRP")[0].compareTo("Sim")
== 0 ? true : false;
        String unidade = request.getParameterValues("unidade")[0];
        double custo = Double.parseDouble(request.getParameterValues("custo")[0]);
```

```
        double qtdeLote =
Double.parseDouble(request.getParameterValues("qtdeLote")[0]);
        double leadTime =
Double.parseDouble(request.getParameterValues("leadTime")[0]);
        //int LLC = request.getParameterValues("LLC")[1];
        double estSeguranca =
Double.parseDouble(request.getParameterValues("estSeguranca")[0]);

        Context ctx = new InitialContext();
        ManipItensHome manipHome = (ManipItensHome) ctx.lookup(
            "java:comp/env/ejb/MRP/ManipItens"
        );
        ManipItens manip = manipHome.create();
        ItemBMP item = manip.findByPrimaryKey(codigo);

        item.setDescricao(descricao);
        item.setProdComp(prodComp);
        item.setFinalInter(finalInter);
        item.setRealFant(realFant);
        item.setCalcMRP(calcMRP);
        item.setUnidade(unidade);
        item.setCusto(custo);
        item.setQtdeLote(qtdeLote);
        item.setLeadTime(leadTime);
        item.setEstSeguranca(estSeguranca);
%>
<FORM name='manip' action='itensManip.jsp' method=post>
<INPUT type=hidden name='acao' value=''>
<TABLE border='0' cellpadding='0' cellspacing='0'>
<TR><TD colspan='2' align='center'> </TR>
<TR><TD colspan='2' align='center'><H2 align='center'>Item <%=codigo%> alterado com
sucesso</H2></TR>
<TR>
    <TD width='400' valign='top'>
        <TABLE border='0' cellpadding='0' cellspacing='5'>
            <TR><TD width='200'>Código</TD>
                <TD><INPUT type=text name='codigo' value='<%=codigo%>'></TD>
            </TR>
            <TR><TD width='200'>Descrição</TD>
                <TD >
                    <INPUT type=text name='descricao' value='<%=descricao%>'></TD>
            </TR>
            <TR><TD width='200' >Unidade</TD>
                <TD ><SELECT name='unidade'>
                    <OPTION value=''>Escolha uma unidade de medida
                    <OPTION value='g' <%=(unidade.compareTo("g")==0) ? "selected" :
""%>>g
                    <OPTION value='ml' <%=(unidade.compareTo("ml")==0) ? "selected"
: ""%>>ml
                    <OPTION value='cm' <%=(unidade.compareTo("cm")==0) ? "selected"
: ""%>>cm
                    <OPTION value='unidades' <%=(unidade.compareTo("unidades")==0)
? "selected" : ""%>>unidades
                    </SELECT></TD>
            </TR>
            <TR><TD width='200'>Custo</TD>
                <TD><INPUT type=text name='custo' value='<%=custo%>'></TD>
            </TR>
            <TR><TD width='200'>Quantidade do lote</TD>
                <TD><INPUT type=text name='qtdeLote' value='<%=qtdeLote%>'></TD>
            </TR>
            <TR><TD width='200'>Lead time</TD>
                <TD><INPUT type=text name='leadTime' value='<%=leadTime%>'></TD>
            </TR>
            <TR><TD width='200' >Estoque de Segurança</TD>
                <TD >
```

```
                    <INPUT type=text name='estSeguranca'
value='<%=estSeguranca%>'></TD>
                </TR>
            </TABLE>
        </TD>
        <TD align='top'>
            <TABLE border='1' cellpadding='5' cellspacing='0'>
                <TR><TD width='200'><INPUT type=radio name='prodComp' value='Produzido'
<%=(prodComp.compareTo("Produzido")==0) ? "checked" : ""%>>Produzido<BR>
                    <INPUT type=radio name='prodComp' value='Comprado'
<%=(prodComp.compareTo("Comprado")==0) ? "checked" : ""%>>Comprado</TD>
                </TR>
                <TR><TD width='200'><INPUT type=radio name='finalInter' value='Final'
<%=(finalInter.compareTo("Final")==0) ? "checked" : ""%>>Final<BR>
                    <INPUT type=radio name='finalInter' value='Intermediario'
<%=(finalInter.compareTo("Intermediario")==0) ? "checked" : ""%>>Intermediario</TD>
                </TR>
                <TR><TD><INPUT type=radio name='realFant' value='Real'
<%=(realFant.compareTo("Real")==0) ? "checked" : ""%>>Real<BR>
                    <INPUT type=radio name='realFant' value='Fantasma'
<%=(realFant.compareTo("Fantasma")==0) ? "checked" : ""%>>Fantasma</TD>
                </TR>
                <TR><TD>Calculado com MRP?<BR>
                    <INPUT type=radio name='calcMRP' value='Sim' <%=(calcMRP) ?
"checked" : ""%>>Sim<BR>
                    <INPUT type=radio name='calcMRP' value='Nao' <%=(calcMRP) ? ""
: "checked"%>>Não</TD>
                </TR>
            </TABLE>
        </TD>
</TR></TABLE>
        <TABLE border='0' cellpadding='5' cellspacing='0'>
            <TR><TD><INPUT type=submit name='alterar' value='Alterar'
onclick='document.manip.acao.value="alterar";'>
                </TD>
                <TD><INPUT type=submit name='remover' value='Remover'
onclick='document.manip.acao.value="remover";'>
                </TD>
                <TD><INPUT type=button name='processos' value='Processos'
onclick='document.location.href="processosManip.jsp?acao=busca&codItem=<%=codigo%>&
codRecurso=";'>
                </TD>
                </TD>
            </TR>
        </TABLE>
</FORM>
<%          }     //"try"
            catch (Exception e) {%><%
                out.println("<br>Erro na alteração");
                e.printStackTrace();
            }
        }
        else if (acao.compareTo("remover") == 0) {
            try {
                String codigo = request.getParameterValues("codigo")[0]; // PK
%>
<FORM name='manip' action='itensManip.jsp' method=post>
<INPUT type=hidden name='acao' value='confirma'>
<INPUT type=hidden name='codigo' value='<%=codigo%>'>
<TABLE border='0' cellpadding='0' cellspacing='0'>
    <TR><TD colspan='2' align='center'> </TR>
    <TR><TD colspan='2' align='center'><H2 align='center'>Por favor confirme a
exclusão do item <%=codigo%></H2></TR>
    <TR><TD><INPUT type=submit name='confirma' value='Confirmar'></TD></TR>
</TABLE>
</FORM>
```

```
<%      }    //"try"
        catch (Exception e) {%><%
            out.println("<br>Erro na remoção");
            e.printStackTrace();
        }
    }
    else if (acao.compareTo("confirma") == 0) {
        try {
            String codigo = request.getParameterValues("codigo")[0]; // PK
            Context ctx = new InitialContext();
            ManipItensHome manipHome = (ManipItensHome) ctx.lookup(
                "java:comp/env/ejb/MRP/ManipItens"
            );
            ManipItens manip = manipHome.create();
            ItemBMP item = manip.findByPrimaryKey(codigo);
            item.remove();
%>
<TABLE border='0' cellpadding='0' cellspacing='0'>
    <TR><TD colspan='2' align='center'> </TR>
    <TR><TD colspan='2' align='center'><H2 align='center'>Item <%=codigo%> removido
com sucesso</H2></TR>
</TABLE>
<%        }    //"try"
        catch (Exception e) {%><%
            out.println("<br>Erro na remoção");
            e.printStackTrace();
        }
    }
}
%>

<%-- <jsp:getProperty name="beanInstanceName"  property="propertyName" /> --%>

</body>
</html>
```

codigo 6. ItensManip.jsp

Esse código JSP realiza todos os requisitos propostos para a manipulação dos itens de estoque .

A criação de itens é feita conforme formulário da Figura 11. Essa criação pode ser acessada diretamente pelo menu à esquerda da aplicação em "Criar novo item". Esse atalho faz uma requisição ao JSP com o parâmetro "acao" igual a "inserir".

Como pode ser visto pelo codigo 6 essa requisição é respondida com o formulário de inserção e, quando recebida juntamente com os dados do formulário, cria um novo EntityBean com os dados fornecidos.

É interessante notar aqui, já que temos o código de exemplo, como é criado o EJB de acordo com o funcionamento esquematizado na Figura 8. Primeiramente é encontrada (via JNDI) a interface Home. Por meio de uma instância dessa interface é então criado um novo Item (lembrando que a Home funciona como uma fábrica para os objetos).

Através do menu pode ser acessada também a pesquisa de itens. Esse atalho leva ao formulário mostrado na Figura 27. No caso de uma busca por descrição igual a "cor" obteve-

se a listagem mostrada na mesma figura à direita. E ao se clicar no primeiro item listado (Corante Azul) obteve-se o resultado no formulário logo abaixo. Se a pesquisa tivesse iniciado diretamente com o código do item (corante01) no formulário inicial chegar-se-ia diretamente ao formulário final.

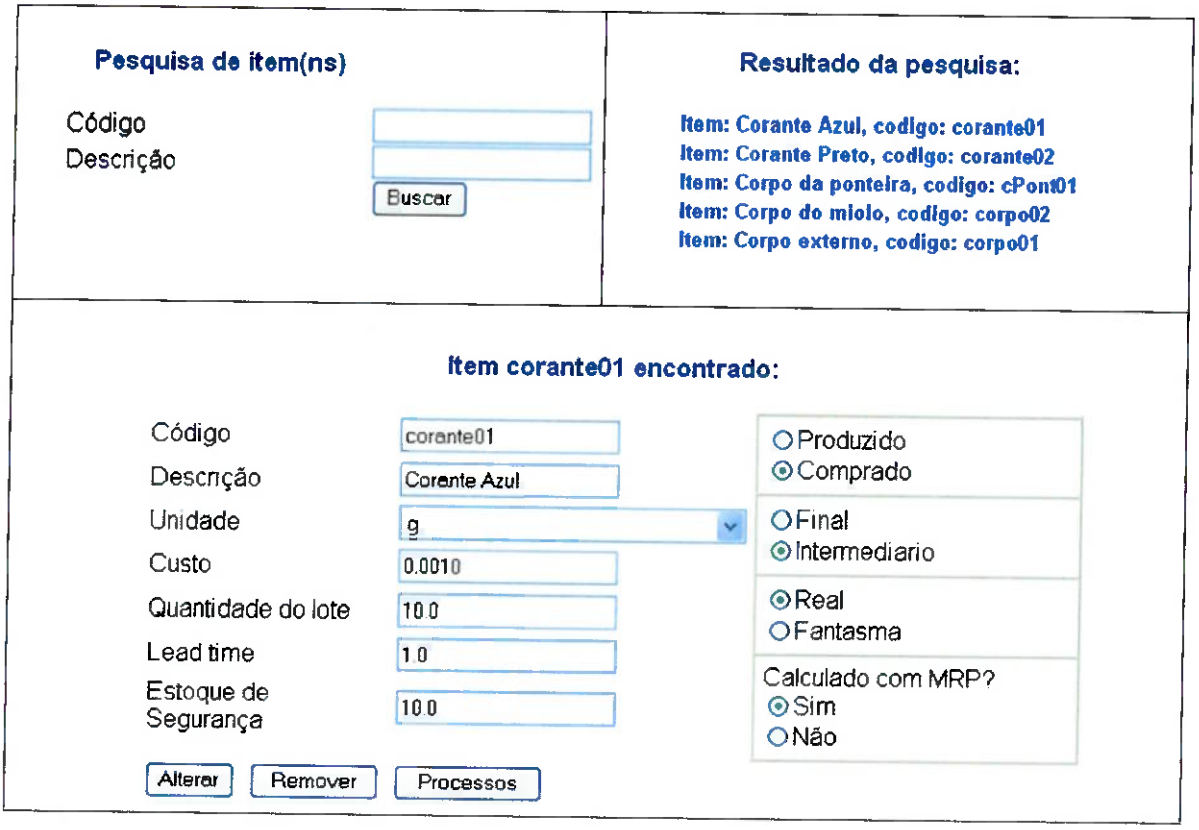


Figura 27. Exemplo de busca de itens.

Esse formulário, como pode ser visto, permite a alteração e remoção do item. Permite também que sejam consultados os processos cadastrados para esse item.

Aqui podemos notar também, consultando o codigo 6, a forma como o objeto é acessado de acordo com o esquema dado na Figura 7. Primeiramente é criada uma instância da interface Home e, através desta, é acessado o Entity Bean em si com a criação de uma outra instância para o mesmo.

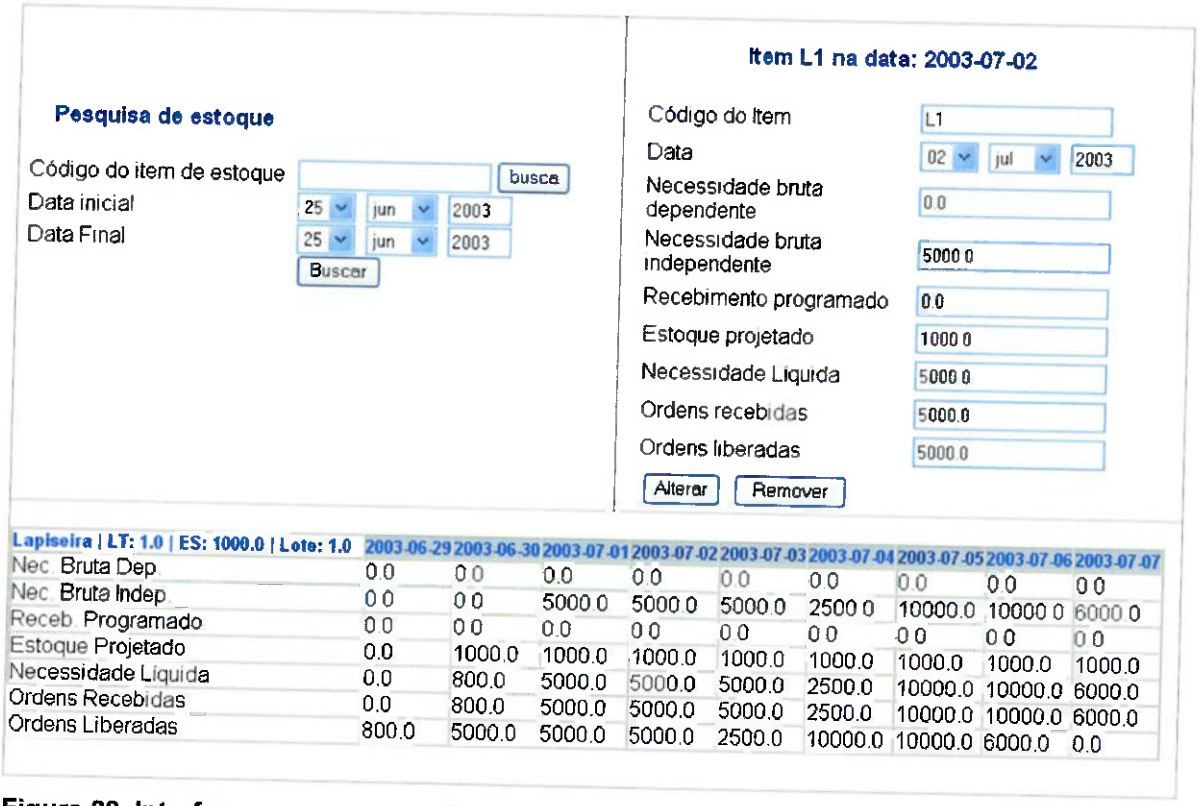
5.2.2 Estoque

Para o estoque também existe um Entity Bean e uma página JSP (código em anexo). A definição de estoque (para um item em uma data) é feita conforme interface na Figura 12.

Normalmente essa definição só precisa ser feita para definir a demanda independente dos itens finais (que serão vendidos). Ou seja, na verdade essa definição de estoque faz o papel do MPS (*Master Production Schedule*) para o qual não existe um módulo específico no sistema (ver item 2.2.3 para uma explicação). O funcionamento cliente e a comunicação

com os componentes servidores funciona de forma similar ao já discutido para os itens (ver código em anexo para detalhes).

O menu da aplicação permite também a realização da pesquisa em estoque de acordo com o formulário da Figura 28 acima à esquerda. No caso de uma busca para o item Lapiseira no intervalo de 25 de junho a 25 de julho, o resultado está mostrado na mesma figura mais abaixo. São mostradas apenas as datas para as quais existe demanda ou ordens definidas.



Lapiseira | LT: 1.0 | ES: 1000.0 | Lote: 1.0	2003-06-29	2003-06-30	2003-07-01	2003-07-02	2003-07-03	2003-07-04	2003-07-05	2003-07-06	2003-07-07
Nec. Bruta Dep.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Nec. Bruta Indep.	0.0	0.0	5000.0	5000.0	5000.0	2500.0	10000.0	10000.0	6000.0
Receb. Programado	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Estoque Projetado	0.0	1000.0	1000.0	1000.0	1000.0	1000.0	1000.0	1000.0	1000.0
Necessidade Liquida	0.0	800.0	5000.0	5000.0	5000.0	2500.0	10000.0	10000.0	6000.0
Ordens Recebidas	0.0	800.0	5000.0	5000.0	5000.0	2500.0	10000.0	10000.0	6000.0
Ordens Liberadas	800.0	5000.0	5000.0	5000.0	2500.0	10000.0	10000.0	6000.0	0.0

Figura 28. Interface para pesquisa de estoque

Ao clicar no item (na mesma tabela acima à esquerda) o usuário tem a possibilidade de editá-lo indo diretamente para o formulário mostrado no item anterior. Ao clicar em alguma das datas no cabeçalho da tabela o usuário é encaminhado para outro formulário para edição ou remoção do registro de estoque como mostra a Figura 28 acima à direita (No caso foi clicada a data 2003-07-02).

O funcionamento do código para a realização de consultas, alteração e remoção também é similar ao dos itens (ver código anexo para detalhes).

5.3 BOM – Estrutura de materiais

A manipulação da estrutura de materiais é feita de forma semelhante ao dos *Entity Beans* anteriormente apresentados. Existe no entanto, além do *Entity Bean* servidor e da página JSP cliente, um *Session Bean* que faz a manipulação da árvore.

Isso é necessário em função da maior complexidade para a manipulação da estrutura hierárquica da árvore. Na verdade o padrão sugerido para trabalhar com EJBs é sempre o de acrescentar um Session Bean como camada intermediária entre os Entity Beans e o cliente.

Assim a página JSP acessa preferivelmente o Session Bean e este conta com métodos para manipular a estrutura de materiais.

Além disso, outra diferença desse Entity Bean é a presença de duas tabelas no banco para realizar a persistência ao invés de uma só como nos demais. Isso ocorre porque é necessário persistir a estrutura da árvore além dos itens isoladamente. Como estamos trabalhando com BMP (*Bean Managed Persistence*) essa persistência pode ser feita com os métodos do próprio Entity Bean tomando-se o cuidado de realizar inserções, remoções e alterações em ambas as tabelas quando a árvore é alterada[4].

O menu da aplicação fornece dois atalhos para a manipulação da lista de materiais: a criação de uma nova árvore e a consulta de árvores existentes. A criação de uma nova árvore é feita através do formulário da Figura 15.

O outro atalho fornecido pelo menu é o de pesquisa da BOM e é de onde se inicia toda a manipulação das listas de materiais. Ao clicar na pesquisa o usuário recebe um formulário simples com apenas uma caixa de seleção (Figura 29 acima à esquerda) onde ele pode escolher entre as estruturas existentes.

[4] Para maiores detalhes queira referenciar os códigos em anexo

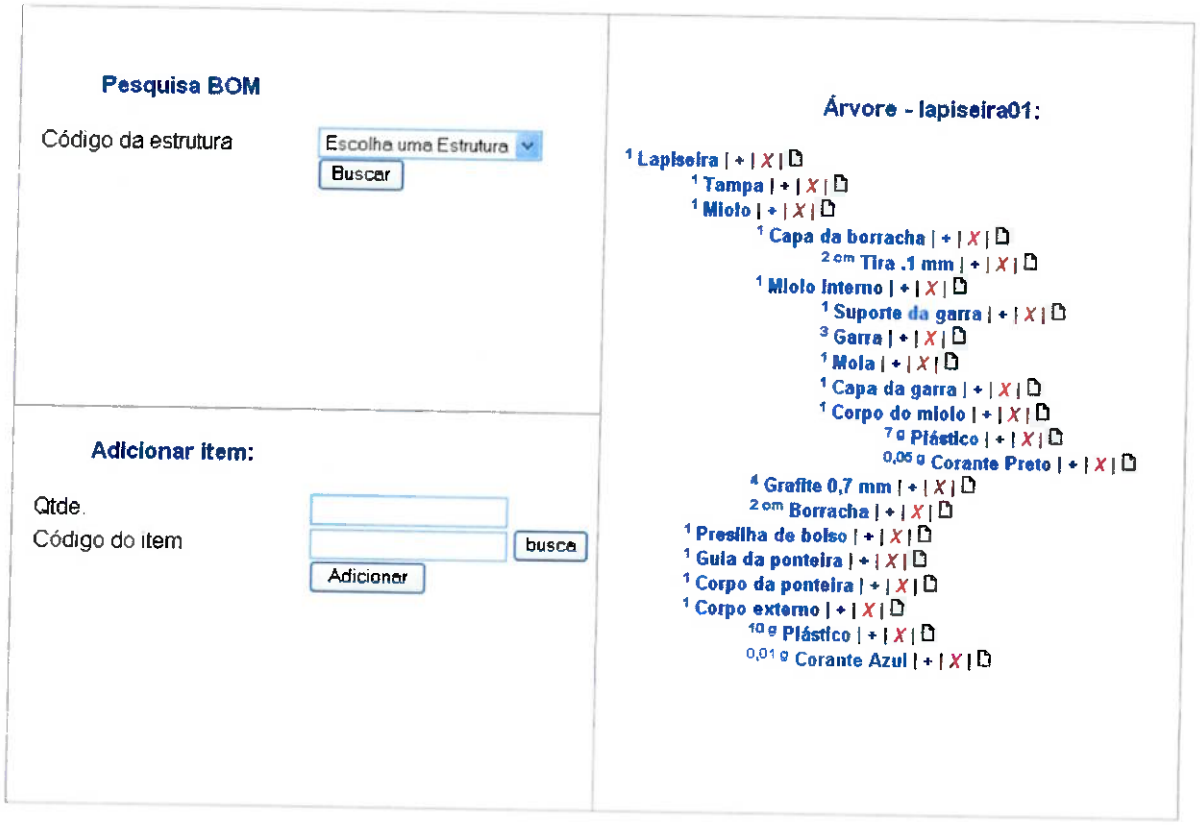


Figura 29. Pesquisa da lista de materiais - BOM

Ao escolher alguma é retornada a árvore como mostrada na mesma figura à direita. A estrutura é mostrada com cada nível indentado de determinado espaço, mostrando os itens e as respectivas quantidades. Para cada item na árvore o usuário tem a possibilidade de realizar as seguintes ações:

- Clicando no nome do item é acessado diretamente o formulário para edição de itens apresentado no item 5.2.1.
- Clicando no sinal de "+" pode-se adicionar outro item abaixo do item na linha clicada. Para isso o usuário recebe outro formulário mostrado na mesma Figura 29 abaixo à esquerda.
- Clicando no sinal de "X" o item e toda a árvore abaixo do mesmo é removida (como em todos os outros módulos a remoção deve ser precedida por um aviso e confirmação do usuário).
- Clicando no símbolo à direita de cada linha pode-se consultar diretamente o estoque de hoje a um mês para o item.

5.4 Processos

O código e funcionamento geral para recursos e processos é semelhante ao dos outros módulos como de estoque.

5.4.1 Recursos

O menu oferece atalhos para criação e pesquisa de recursos. Para criar é utilizado o formulário da Figura 17.

Ao clicar em pesquisa o usuário recebe o formulário da Figura 30 (acima). Como os recursos existem em menor quantidade a pesquisa pode ser feita diretamente em uma caixa de seleção (o mesmo é utilizado para escolha de recurso na definição e pesquisa de processos e capacidade como será visto mais adiante).

Com a escolha de algum recurso é devolvido ao usuário o formulário da mesma figura (abaixo) onde é possivel alterar e remover o recurso além de realizar dois tipos de consulta diretamente. A primeira é de todos os processos que utilizam esse recurso. A segunda é uma pesquisa de capacidade (que está inclusive presente no menu de capacidade como será visto adiante) que retorna todas as datas onde ocorre carência desse recurso (conforme calculadas pelo CRP) para realizar a produção programada pelo MRP (ver item 2 para explicação dos conceitos).

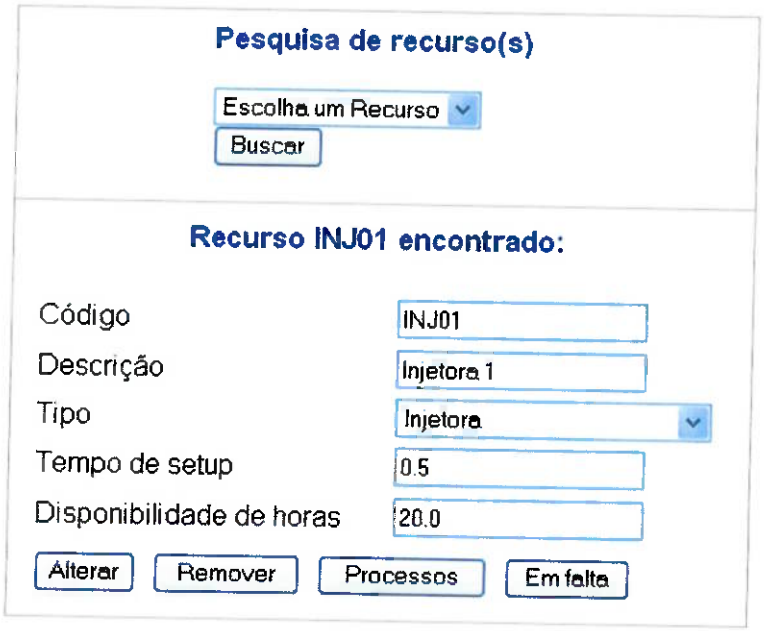


Figura 30. Pesquisa de recursos

5.4.2 Processos

O menu oferece dois atalhos: criar e pesquisar processos. Para criar é utilizado o formulário da Figura 18.

Para pesquisa é utilizado o formulário na Figura 31 acima à direita. Especificando um recurso a consulta retorna o resultado no quadro à esquerda no centro e para um item no quadro mais abaixo. Ao se clicar em qualquer uma das linhas listadas o usuário recebe um formulário como o da figura à direita onde ele tem a possibilidade de alterar ou remover o processo.

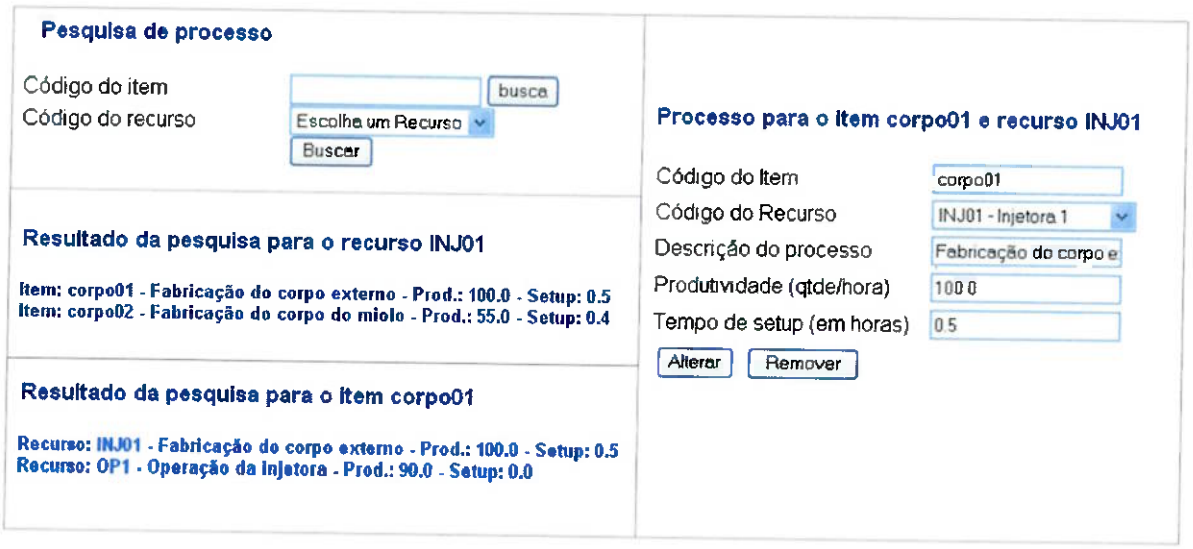


Figura 31. Pesquisa de processos

5.5 Capacidade

O módulo de capacidade funciona de forma semelhante ao de estoque. Existe um *Entity Bean* na camada de servidor e uma página JSP na camada do cliente. Existe no entanto um *Session Bean* para complementar a camado do servidor assim como na BOM.

A definição de capacidade é feita através do formulário na Figura 20. Como a disponibilidade de um recurso não tende a mudar muito constantemente a definição é feita para um intervalo de tempo. Ao se definir a capacidade é retornada uma consulta para o período definido conforme será mostrado a seguir.

A pesquisa é feita pelo formulário acima à direita na Figura 32.

INJ01	2003-06-21	2003-06-22	2003-06-23	2003-06-24	2003-06-25	2003-06-26	2003-06-27
Necessidade em horas	0	0	0	0	105,8545	91,3091	91,3091
Necessidade em qtde.	0	0	0	0	6	5	5
Horas disponíveis	120	120	120	120	120	120	120
Qtde. disponível	6	6	6	6	6	6	6
Necessidade líquida em horas	0	0	0	0	-14,1455	-28,6909	-28,6909
Necessidade líquida em qtde.	0	0	0	0	0	-1	-1

Figura 32. Pesquisa de capacidade.

O resultado é a tabela na mesma figura abaixo. Ao clicar no recurso (acima à esquerda na tabela) o usuário recebe diretamente o formulário para edição do recurso. Ao clicar em qualquer uma das datas é retornado o formulário da mesma figura (acima à direita) que permite a alteração da quantidade disponível (os outros campos estão desabilitados e só podem ser alterados pelo CRP) e a remoção do registro.

5.6 MRP / CRP

Para realização do processamento do CRP e MRP existe um item chamado "Controle" no menu. Esse item abre a interface da Figura 33 à esquerda que permite processar o MRP e o CRP. Para ambos é retornada uma tela que solicita que o usuário espere o fim do processamento, quando é completada a informação de sucesso.

No caso do CRP no fim do processamento é acrescentado um formulário para realizar uma busca nos recursos que estarão em falta de acordo com o processamento.

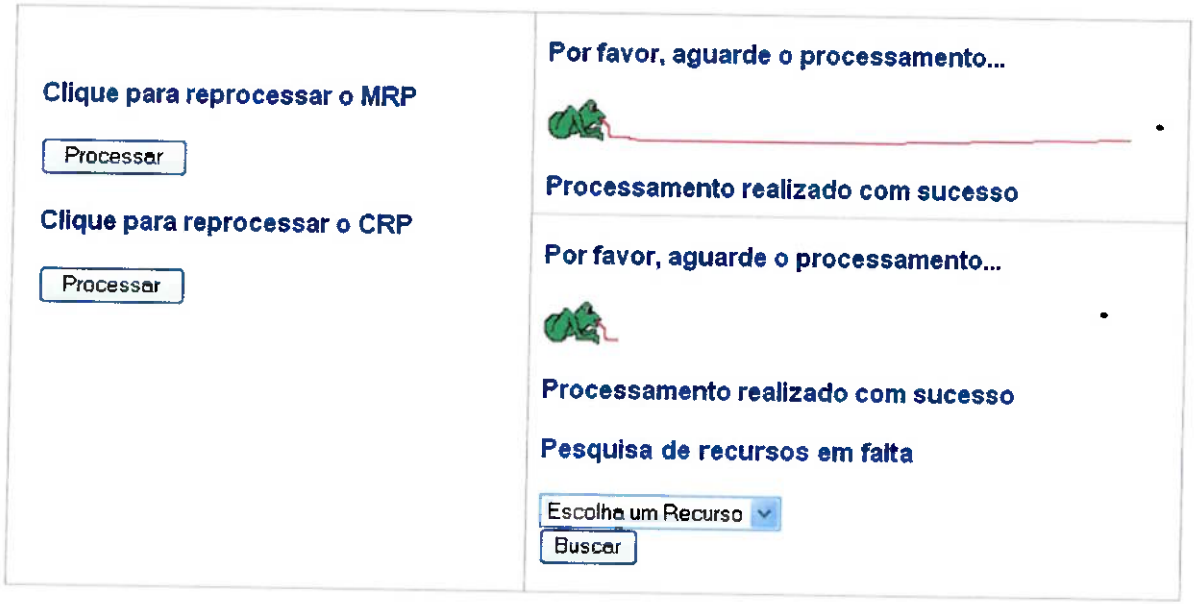


Figura 33. Controle de processamento no sistema

5.7 Preparação do ambiente

A preparação do ambiente de desenvolvimento e execução do sistema foi feita seguindo-se os seguintes passos:

- Instalação dos kits de desenvolvimento java
  - **j2sdk1.4.0** – Kit de desenvolvimento padrão com as APIs (Application Program Interfaces) mais comuns para a programação java. A versão utilizada é 1.4.0.
  - **j2sdkee1.3.1** – Kit de desenvolvimento "Enterprise Environment" com outras API pertencentes à plataforma J2EE e com a opção de um servidor de aplicações (que não será utilizado, já que utilizaremos o JBoss).
- Instalação do ambiente de desenvolvimento (IDE – Integrated Development Environment) **Forte for Java** (ou Sun ONE).
- Instalação do servidor de aplicações **Jboss**
- Instalação da ferramenta de compilação **Ant**
- Instalação da ferramenta complementar para geração de arquivos **XDocLet**.

6 Conclusão

Esse projeto envolveu duas área de conhecimento bem definidas.

A primeira diz respeito a **planejamento e controle da produção**. No capítulo 2 desse relatório foram discutidos conceitos de MRP, CRP, MRPII e ERP, sua origem, funcionamento e a forma como esses conceitos são utilizados pelas empresas.

A segunda diz respeito a **tecnologia para desenvolvimento de software**. O capítulo 3 discute de forma relativamente completa a linguagem java, sua evolução, o nascimento da plataforma J2EE, seu funcionamento geral e suas vantagens. Além disso é dada ênfase especial à modelagem de sistemas.

Com relação à primeira, esse trabalho está longe de ser muito abrangente ou inovador. No entanto acreditamos que o fato de unir esse tópico ao outro referido e propor uma modelagem orientada a objetos para um sistema do gênero torna esse projeto um candidato a contribuir de forma não despresível para o campo de engenharia de produção.

Com relação à segunda, acreditamos que o projeto tenha bastante de inovador. Não só porque a tecnologia utilizada é relativamente recente, mas também pelo fato de unir os conceitos acima relacionados e por dar ênfase especial na modelagem orientada a objetos (e especialmente orientada aos requisitos de projeto).

Por fim consideramos que o projeto tem um valor especial no sentido didático (que inclusive consta do seu nome) já que, acreditamos, um estudante que queira se iniciar no uso da plataforma J2EE encontrará no resultado desse projeto fonte valiosa de informação (incluíndo diversos exemplos para uma aplicação relativamente complexa).

6.1 Recomendações

Para dar continuidade a esse projeto podem ser dadas recomendações distintas de acordo com o interesse específico do projetista.

Com relação ao sistema MRPII em si podem ser adicionados diversos módulos marginais, inclusive integrando com outros módulos na empresa (como contabilidade, vendas, engenharia etc.).A vantagem do uso de componentes é que cada módulo pode ser independente com relação ao seu desenvolvimento como discutido anteriormente. Podem inclusive ser utilizados módulos específicos desse projeto para integrar uma solução já parcialmente desenvolvida. Com relação à tecnologia para desenvolvimento de software recomenda-se um avanço no entendimento da plataforma J2EE com uso de outras possibilidades como conectividade com sistemas legados, uso de transações e, principalmente, tratamento de segurança da aplicação.

7 Bibliografia

ANT documentation - http://ant.apache.org/index.html (link válido em 24/06/2002).

BARRELA, Wagner Däumichen. Sistemas especialistas modulados e abrangentes para a gestão de operações. São Paulo: EPUSP; Tese de doutorado, 2000. 179 p.

BODOFF, Stephanie *et. al. The J2EE Tutorial*. Boston: Addison-Wesley, 2002. 517 pp.

BOOCH, Grady. *Object-oriented analysis and design with applications*. 2 ed. Redwood: Benjamin/Cummings, 1994. 589 pp.

BOOCH, Grady; JACOBSON, Ivar; RUMBAUGH, James. *The Unified Modeling Language User Guide*. 1 ed. Boston: Addison-Wesley, 1999. 482 pp.

CORRÊA, Henrique L. & GIANESI, Irineu G. N. *Just in Time, MRPII e OPT*: um enfoque estratégico. 2 ed. São Paulo: Atlas, 1996. 186 pp.

ECKEL, Bruce. *Thinking in Java*. 2 ed, 2000. http://www.bruceeckel.com (link válido em 24/06/2002).

FAVARETTO, Fábio. Uma contribuição ao processo de gestão da produção pelo uso da coleta automática de dados do chão de fábrica. São Carlos: USP, Tese de doutorado, 2001. 235 pp.

FIGUEIREDO, Fernando M. Desenvolvimento de um MRP didático. São Paulo EPUSP; Projeto de graduação, 2001. 118 pp.

FULLMANN, Claudiney et. al. *MRP/MRPII, MRPIII (MRP+JIT+KANBAN), OPT e GDR*. 1 ed. São Paulo: IMAM, 1989. 284.

GAMMA, Erich et. al. *Design Patterns*: elements of reusable object oriented software. 1 ed. Boston: Addison-Wesley, 1995. 395 pp.

HANNA, Phil. *JSP*: the complete reference. 1 ed. Berkeley: McGraw Hill, 2001. 876 pp.

MARINESCU, Floyd. *EJB Design Patterns*: advanced patterns, processes and idioms. 1 ed. New York: John Wiley & Sons, 2002. 289 pp.

ODEN et al. *Handbook of Material and Capacity Requirements Planning*. 1 ed. New York: McGraw Hill, 1993. 458 pp.

ORFALI, Robert et. al. *Client/Server Survival Guide*. 3 ed. New York: Jonh Wiley & Sons, 1999. 762 pp.

ROMAN, Ed. *et. al. Mastering Enterprise JavaBeans™*. 2 ed. New York: John Wiley & Sons, 2002. 672 pp.

ROSENBERG, Doug & SCOTT, Kendall. *Use case driven object modeling with UML*: a practical approach. 1 ed. Reading: Addison-Wesley, 1999. 165 pp.

SÁ, Guilherme F. J. de. Sistema SCADA em CORBA / JAVA. São Paulo EPUSP; Projeto de graduação, 1999. 165 pp.

SUN MICROSYSTEMS. The Java 2 Enterprise Edition Developer's Guide. Palo Alto, 2000. http://java.sun.com (link válido em 24/06/2002).

UML documentation – http://www.uml.org (link válido em 24/06/2002).

XDOCLET documentation - http://www.xdoclet.com/ (link válido em 24/06/2002).